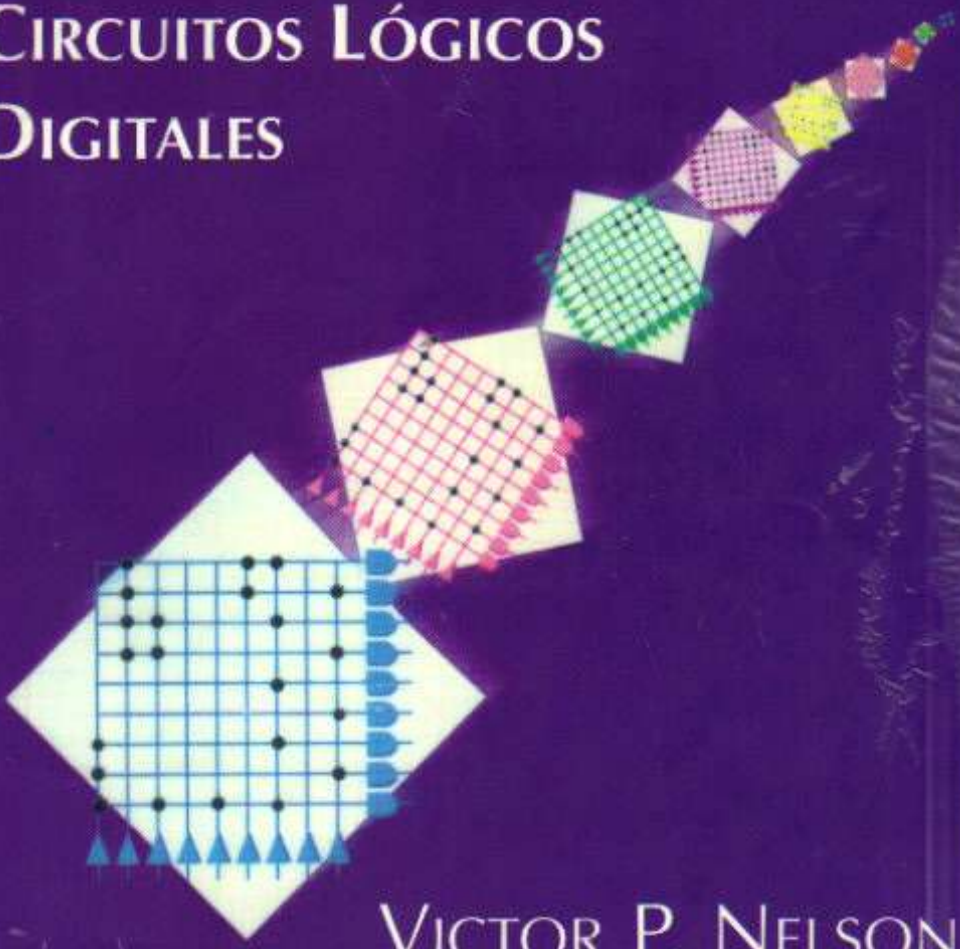
Análisis y Diseño de Circuitos Lógicos Digitales



Victor P. Nelson
H. Troy Nagle
Bill D. Carroll
J. David Irwin

PEARSON
Prentice Hall

621.395
A532

Análisis y diseño de circuitos lógicos digitales

Primera edición

UNIVERSIDAD TECNOLOGICA DE PERE
BIBLIOTECA

Victor P. Nelson
Auburn University

H. Troy Nagle
North Carolina State University

Bill D. Carroll
University of Texas-Arlington

J. David Irwin
Auburn University

Traducción:

M en C, Oscar Alfredo Palmas Velasco
Facultad de Ciencias, UNAM

Revisor técnico

Carlos Henández Pérez
Ingeniero Mecánico Electricista
UNAM

$71.655
9 DE SEPTIEMBRE DE 2003
GRUPO NORMA LTDA

Biblioteca Univ. Tecnologica de Pereira
6 3100 00045511 5

PRENTICE-HALL HISPANOAMERICANA, S.A.

MÉXICO • NUEVA YORK • BOGOTÁ • LONDRES • SYDNEY • PARÍS • MUNICH • TORONTO • NUEVA DELHI • TOKIO • SINGAPUR • RÍO DE JANEIRO • ZURICH

Contenido

Prefacio XVII

0 Introducción 1

0.1 Historia de la computación 1

0.1.1 Los inicios: Computadoras mecánicas 1

0.1.2 Primeras computadoras electrónicas 2

0.1.3 Las primeras cuatro generaciones de computadoras 2

0.1.4 La quinta generación y más allá 4

0.2 Sistemas digitales 4

0.2.1 Sistemas digitales y analógicos 4

0.2.2 Jerarquía de diseño de un sistema digital 5

0.3 Organización de una computadora digital de programa almacenado 12

0.3.1 Instrucciones de la computadora 13

0.3.2 Representación de la información en las computadoras 13

0.3.3 Hardware de computadora 15

0.3.4 Software de computadora 17

0.4 Resumen 19

1 Sistemas numéricos y códigos 20

1.1 Sistemas numéricos 21

1.1.1 Notación posicional 21

1.1.2 Sistemas numéricos de uso común 22

1.2 Aritmética 22

1.2.1 Aritmética binaria 23

1.2.2 Aritmética octal 27

1.2.3 Aritmética hexadecimal 28

1.3 Conversiones de base 30

1.3.1 Métodos de conversión 31

1.3.2 Algoritmos generales de conversión 35

1.3.3 Conversión entre la base A y la base B cuando $B=A^k$ 36

1.4 Representación de números con signo 37

1.4.1 Números con magnitud y signo 38

1.4.2 Sistemas numéricos complementarios 38

1.5 Códigos de computadora 55

1.5.1 Códigos numéricos 55

1.5.2 Códigos de caracteres y otros códigos 61

1.5.3 Códigos para detección y corrección de errores 65

1.6 Resumen 73

2 Métodos algebraicos para el análisis y síntesis de circuitos lógicos 78

2.1 Fundamentos del álgebra booleana 79

2.1.1 Postulados básicos 79

2.1.2 Diagramas de Venn para los postulados [2] 80

2.1.3 Dualidad 81

2.1.4 Teoremas fundamentales del álgebra booleana 84

2.2 Funciones de conmutación 91

2.2.1 Tablas de verdad 93

2.2.2 Formas algebraicas de las funciones de conmutación 94

2.2.3 Deducción de formas canónicas 101

2.2.4 Funciones con especificación incompleta 103

2.3 Circuitos de conmutación 104
2.3.1 Compuertas lógicas electrónicas 105
2.3.2 Componentes funcionales básicos 108
2.4 Análisis de circuitos combinatorios 120
2.4.1 Método algebraico 120
2.4.2 Análisis de diagramas de tiempos 123
2.5 Síntesis de circuitos lógicos combinatorios 128
2.5.1 Redes AND-OR y NAND 128
2.5.2 Redes OR-AND y NOR 130
2.5.3 Circuitos de dos niveles 131
2.5.4 Circuitos AND-OR-Inversor 133
2.5.5 Factorización 134
2.6 Aplicaciones 136
2.7 Diseño de circuitos lógicos asistido por computadora 140
2.7.1 El ciclo de diseño 140
2.7.2 Modelado de un circuito digital 140
2.7.3 Síntesis de diseño y herramientas de captura 148
2.7.4 Simulación lógica 152
2.8 Resumen 165

3 Simplificación de las funciones de conmutación 172
3.1 Objetivos de la simplificación 173
3.2 Características de los métodos de minimización 174
3.3 Mapas de Karnaugh 175
3.3.1 Relación con los diagramas de Venn y las tablas de verdad 176
3.3.2 Mapas K de cuatro o más variables 17

3.4 Trazo de funciones en forma canónica sobre el mapa K 179

3.5 Simplificación de las funciones de conmutación mediante los mapas K 185

3.5.1 Criterios para la simplificación de funciones mediante mapas K 187

3.5.2 Terminología general para la minimización de una función de conmutación 187

3.5.3 Algoritmos para la deducción de formas SOP minimales a partir de mapas K 188

3.6 Forma POS mediante mapas K 197

3.6.1 Terminología general para las formas POS 197

3.6.2 Algoritmos para la deducción de formas POS minimales a partir de mapas K 197

3.7 Funciones con especificación incompleta 203

3.8 Uso de mapas K para eliminar riesgos de tiempo 206

3.9 Método de minimización tabular de Quine-McCluskey 211

3.9.1 Procedimiento de cubierta 215

3.9.2 Funciones con especificación incompleta 218

3.9.3 Sistemas con múltiples salidas 220

3.10 Algoritmo de Petrick 222

3.11 Mínimización de funciones de conmutación asistida por computadora 225

3.11.1 Representación cúbica de las funciones de conmutación 227

3.11.2 Métodos algebraicos para determinar implicantes primos 228

3.11.3 Identificación de implicantes primos esenciales 230

3.11.4 Cómo completar una cubierta minima 231

3.11.5 Otros algoritmos de minimización 234

3.12 Resumen 235

4 Lógica combinatoria modular 242

4.1 Diseño modular descendente 243

4.2 Decodificadores 245

4.2.1 Estructura de circuitos decodificadores 246

4.2.2 Implantación de funciones lógicas mediante decodificadores 247

4.2.3 Entradas para control de activación 249

4.2.4 Decodificadores MSI estándar 252

4.2.5 Aplicaciones de los decodificadores 253

4.3 Codificadores 259

4.3.1 Estructuras de un circuito codificador 260

4.3.2 Codificadores MSI estándar 264

4.4 Multiplexores/selectores de datos 268

4.4.1 Estructuras de circuitos multiplexores 268

4.4.2 Multiplexores MSI estándar 270

4.4.3 Aplicaciones de los multiplexores 277

4.5 Demultiplexores/distribuidores de datos 280

4.6 Elementos de aritmética binaria 283

4.6.1 Circuitos sumadores binarios básicos 283

4.6.2 Módulos de sumador binario MSI 285

4.6.3 Unidades sumadoras de alta velocidad 289

4.6.4 Circuitos de resta binaria 294

4.6.5 Detección de un desbordamiento aritmético 295

4.7 Comparadores 298

4.8 Ejemplo de diseño: Una unidad aritmética-lógica de computadora 302

4.9 Diseño de sistemas modulares asistido por computadora 312

4.9.1 Bibliotecas de diseño 312

4.9.2 Trazo de esquemas jerárquicos 314

4.10 Simulación de sistemas jerárquicos 317

4.11 Resumen 319

5 Diseño de un circuito combinatorio con dispositivos lógicos programables 328

5.1 Dispositivos lógicos semiadaptados 329

5.2 Circuitos de arreglos lógicos 330

5.2.1 Operación de diodos en circuitos digitales 330

5.2.2 Arreglos lógicos AND y OR 332

5.2.3 Arreglos AND-OR de dos niveles 333

5.2.4 Arreglos AND y OR programables mediante campos 338

5.2.5 Opciones de polaridad de salida 341

5.2.6 Pines bidireccionales y líneas de retroalimentación 343

5.2.7 Dispositivos comerciales 345

5.3 Arreglos lógicos programables mediante campos 347

5.3.1 Estructuras de circuitos FPLA 347

5.3.2 Realización de funciones lógicas con FPLA 347

5.4 Memoria programable exclusiva para lectura 350

5.4.1 Estructuras de circuitos PROM 350

5.4.2 Realización de funciones lógicas con PROM 352

5.4.3 Tablas de búsqueda 358

5.4.4 Aplicaciones generales de la memoria exclusiva para lectura 360

5.4.5 Tecnologías de memorias exclusivas para lectura 361

5.5 Lógica de arreglos programables 362

5.5.1 Estructuras de circuitos PAL 362

5.5.2 Realización de funciones lógicas con PAL 363

5.5.3 Opciones de salida y retroalimentación de PAL 366

5.6 Herramientas de diseño asistido por computadora para el diseño PLD 371
5.6.1 Representación de diseño con PDL 373
5.6.2 Procesamiento de un archivo de diseño PDL 378
5.7 Resumen 380

6 Introducción a los dispositivos secuenciales 382
6.1 Modelos para circuitos secuenciales 383
6.1.1 Representación de diagrama de bloques 383
6.1.2 Tablas y diagramas de estados 385
6.2 Dispositivos de memoria 387
6.3 Latches 389
6.3.1 Latch set-reset 389
6.3.2 Latch SR con compuertas 396
6.3.3 Latch con retardo 398
6.4 Flip-flops 403
6.4.1 Flip-flops SR maestro-esclavo 404
6.4.2 Flip-flops D maestro-esclavo 406
6.4.3 Flip-flops JK maestro-esclavo 407
6.4.4 Flip-flops D disparados por transición 409
6.4.5 Flip-flops JK disparados por transición 413
6.4.6 Flip-flops T 415
6.4.7 Resumen de latches y flip-flops 417
6.5 Otros dispositivos de memoria 418
6.6 Circuitos de tiempo 418
6.6.1 One-shots 418
6.6.2 El módulo temporizador 555 418
6.7 Creación rápida de prototipos de circuitos secuenciales 421
6.8 Resumen 425

7 Lógica secuencial modular 432

7.1 Registros de corrimiento 433

7.1.1 Un registro de corrimiento genérico 434

7.1.2 Módulos estándar TTL de registro de corrimiento 436

7.2 Ejemplos de diseño con registros 448

7.2.1 Unidad sumadora serial 448

7.2.2 Acumuladores seriales 449

7.2.3 Acumuladores paralelos 449

7.3 Contadores 449

7.3.1 Contadores binarios síncronos 451

7.3.2 Contadores binarios asíncronos 455

7.3.3 Contadores hacia abajo 460

7.3.4 Contadores hacia arriba/hacia abajo 461

7.4 Contadores módulo N 464

7.4.1 Contadores BCD síncronos 464

7.4.2 Contadores BCD asíncronos 467

7.4.3 Contadores módulo 6 y módulo 12 471

7.4.4 Contadores módulo N con reset asíncrono 475

7.4.5 Contadores módulo N con reset síncrono 477

7.5 Registros de corrimiento como contadores 477

7.5.1 Contadores de anillo 478

7.5.2 Contadores de anillo trenzado 483

7.6 Contadores de secuencia múltiple 489

7.7 Multiplicadores de razón fraccionaria digital 490

7.7.1 Módulos TTL 492

7.7.2 Multiplicadores de razón fraccionaria digital en cascada 496

7.8 Resumen 496

8 Análisis y síntesis de circuitos secuenciales síncronos 502

8.1 Modelos de circuitos secuenciales síncronos 503

8.1.1 Modelo de Mealy 504

8.1.2 Modelo de Moore 505

8.2 Análisis de un circuito secuencial 507

8.2.1 Análisis de los diagramas de estados de un circuito secuencial 507

8.2.2 Análisis de los diagramas lógicos de un circuito secuencial 508

8.2.3 Resumen 517

8.3 Síntesis de un circuito secuencial síncrono 519

8.3.1 Procedimiento de síntesis 520

8.3.2 Tablas de entrada de un flip-flop 522

8.3.3 Método de la ecuación de aplicación para los flip-flops JK 524

8.3.4 Ejemplos de diseño 526

8.3.5 Diagramas de máquina de estado algorítmico 547

8.3.6 Método de diseño de una máquina de estados finitos one-hot 553

8.4 Circuitos con especificación incompleta 555

8.4.1 Asignación de estados y realización de circuitos 558

8.5 Diseño asistido por computadora de circuitos secuenciales 558

8.5.1 Captura y síntesis del diseño 559

8.5.2 Análisis y verificación del diseño 565

8.6 Resumen 568

9 Simplificación de circuitos secuenciales 576

9.1 Estados redundantes 577

9.1.1 Equivalencia de estados 577

9.1.2 Equivalencia y relaciones de compatibilidad 579

9.2 Reducción de estados en circuitos con especificación completa 579

9.2.1 Inspección 579

9.2.2 Partición 581
9.2.3 Tabla de implicación 584

9.3 Reducción de estados en circuitos con especificación incompleta 588

9.3.1 Compatibilidad de estados 589
9.3.2 Procedimiento de minimización 594

9.4 Métodos para una asignación óptima de estados 602

9.4.1 Asignaciones de estados únicas 603
9.4.2 Criterios para la asignación de estados 605
9.4.3 Partición 614
9.4.4 Asignaciones óptimas de estados 619

9.5 Resumen 620

10 Circuitos secuenciales asíncronos 624

10.1 Tipos de circuitos asíncronos 625

10.2 Análisis de circuitos asíncronos de modo pulso 627

10.3 Síntesis de circuitos de modo pulso 632

10.3.1 Procedimiento de diseño para circuitos de modo pulso 632

10.4 Análisis de circuitos de modo fundamental 641

10.4.1 Introducción 643
10.4.2 Representaciones tabulares 645
10.4.3 Procedimiento de análisis 648

10.5 Síntesis de circuitos de modo fundamental 648

10.5.1 Procedimiento de síntesis 648

10.6 Introducción a las competencias, ciclos y riesgos 659

10.6.1 Competencias y ciclos 660
10.6.2 Cómo evitar condiciones de competencia 663
10.6.3 Asignaciones de estados libres de competencia 664
10.6.4 Riesgos 671
10.6.5 Análisis 673

10.7 Resumen 673

11 Circuitos secuenciales con dispositivos lógicos programables 686

11.1 Dispositivos lógicos programables con registros 687

11.1.1 Secuenciadores lógicos programables mediante campos 691

11.1.2 PAL con registro 696

11.1.3 PLD con macroceldas lógicas programables 700

11.2 Arreglos programables de compuertas 702

11.2.1 Arreglos de celdas lógicas 705

11.2.2 FPGA ACT 713

11.3 Diseño de circuitos secuenciales y selección de dispositivos PLD 715

11.4 Ejemplos de diseño de PLD 717

11.5 Diseño asistido por computadora de PLD secuenciales 723

11.5.1 Representación del diseño de un circuito secuencial mediante PDL 723

11.5.2 Procesamiento de un archivo de diseño en PDL 729

11.6 Resumen 733

12 Verificación de circuitos lógicos y diseño verificable 738

12.1 Verificación de circuitos lógicos digitales 739

12.2 Modelos de fallas 740

12.3 Verificación de un circuito lógico combinatorio 741

12.3.1 Generación de pruebas 742

12.3.2 Fallas no verificables 751

12.3.3 Redes con varias salidas 752

12.3.4 Conjuntos de pruebas para detección de fallas 753

12.3.5 Localización y diagnóstico de fallas 757

12.3.6 Verificación aleatoria 758

12.4 Verificación de circuitos lógicos secuenciales 760

12.5 Diseño para la verificabilidad 763

12.5.1 Diseño de una ruta de rastreo 764

12.6 Autoverificación integrada 768

12.6.1 Generación seudoaleatoria de vectores de prueba 769

12.6.2 Análisis de identificaciones 772

12.6.3 Observador integrado de bloque lógico 775

12.7 Rastreo de frontera en el nivel de sistema y de tarjeta 777

12.8 Resumen 781

13 Ejemplos de diseño 788

13.1 Máquina tragamonedas electrónica 789

13.1.1 Definición del problema 789

13.1.2 Necesidades del sistema y plan de solución 790

13.1.3 Diseño lógico 792

13.2 Sistema de entrada automática sin llave 801

13.2.1 Definición del problema 801

13.2.2 Necesidades del sistema 803

13.2.3 Diseño lógico 805

13.3 Controlador de tráfico de un carril 810

13.3.1 Necesidades del sistema 813

13.3.2 Diseño lógico 815

13.4 Caja registradora de una tienda de abarrotes 821

13.4.1 Necesidades del sistema 823

13.4.2 Diseño lógico 824

Índice 832

Prefacio

La necesidad de este libro

Este texto tiene su origen en una obra anterior, *An Introduction to Computer Logic* (1974), de Nagle, Carroll e Irwin, texto ampliamente adoptado que abarca los fundamentos del análisis y síntesis de los circuitos lógicos secuenciales y combinatorios. La obra original fue apreciada por su claridad y su eficacia en la enseñanza, y a pesar de los rápidos cambios que hubo en el campo a fines de la década de 1970 y principios de la de 1980, el libro fue utilizado durante muchos años después de su fecha de publicación inicial lo que pone de manifiesto el importante hecho de que en la mayor parte del periodo transcurrido desde la publicación de ese libro, el punto de vista educativo general de los cursos introductorios al diseño digital evolucionó lentamente, al tiempo que los principales cambios tecnológicos eran adoptados con rapidez por la industria.

¡Cómo han cambiado las cosas! En fechas recientes, la proliferación astronómica de las aplicaciones de los circuitos digitales y el fenomenal aumento en la complejidad de dichos circuitos han impulsado cambios significativos en los métodos y herramientas utilizados en el diseño digital. Los *chips* de circuitos integrados a gran escala (VLSI) contienen ahora, por lo general, millones de transistores; los métodos de diseño asistido por computadora (CAD), las celdas estándar, los dispositivos lógicos programables y los arreglos de compuertas han permitido el rápido paso del concepto al circuito terminado, con un énfasis cada vez mayor en los diseños modulares jerárquicos que utilizan bibliotecas de celdas estándar y otros módulos de circuitos prediseñados. Hemos desarrollado un texto que abarca todos estos cambios, pero también hemos tenido cuidado de conservar la amplia cobertura de la teoría y los fundamentos.

Un buen ingeniero de diseño digital necesita bases sólidas de teoría fundamental, junto con un conocimiento de los principios de diseño del mundo real. Este texto abarca ambos aspectos. Mantiene la exhaustiva cobertura de la teoría fundamental de su predecesor. Pero, a fin de abordar los aspectos prácticos del diseño, más de la mitad del texto es material nuevo que refleja los numerosos cambios ocurridos en el pasado reciente, incluido el diseño modular, los métodos CAD y el uso de lógica programable, así como aspectos prácticos, tales como las características de tiempo de los dispositivos y los símbolos lógicos estándar.

A quiénes va dirigida esta obra

Este libro sirve para cursos de circuitos lógicos digitales y sistemas digitales para ingenieros y científicos que podrían participar en el diseño de circuitos VLSI, tarjetas de circuitos impresos, módulos de varios circuitos y circuitos de computadoras.

No se requiere ni presupone un conocimiento básico específico de los circuitos electrónicos o los sistemas de cómputo, por lo que el texto es adecuado para un primer curso de sistemas digitales. Sin embargo, el libro contiene suficiente material avanzado y tiene la suficiente profundidad para cubrir las necesidades de estudiantes más adelantados. Este texto ha sido diseñado para que el instructor tenga cierta flexibilidad en la selección de temas, de acuerdo con las necesidades de su curso específico.

El presente texto también es adecuado para el lector que desee utilizar el enfoque autodidacta con el fin de aprender diseño digital, y es útil como referencia para los ingenieros en ejercicio.

Características significativas

Este libro es una obra única que representa el esfuerzo conjunto de los cuatro autores en tres universidades. Además de una revisión exhaustiva coordinada por el editor, el manuscrito fue utilizado en cursos en las tres universidades durante su desarrollo, con una retroalimentación de los estudiantes y profesores que se incorporó en el libro.

Entre las características importantes están:

- Una cobertura sólida de los conceptos y la teoría fundamentales acoplada con métodos prácticos de diseño real
- Un fuerte hincapié en el desarrollo y uso de metodologías sistemáticas de resolución de problemas y de diseño, apoyadas por más de 250 ejemplos resueltos numerados
- Énfasis en la visualización, apoyada por más de 600 ilustraciones
- Numerosos problemas al final de los capítulos con una amplia gama de niveles de dificultad
- Aspectos de CAD integrados en detalle en todo el texto, sin basarse mucho en productos de CAD de proveedores específicos
- Cobertura del diseño modular jerárquico y los módulos estándar de circuitos digitales
- Un capítulo que contiene proyectos de diseño muy completos
- Dos capítulos que describen los dispositivos lógicos programables y sus aplicaciones en la implantación de los circuitos digitales
- Una introducción detallada de la verificación y el diseño para la verificabilidad
- Apoyo a laboratorios de tarjetas y laboratorios de modelado y simulación con base en CAD
- Un manual del maestro con soluciones completas de todos los problemas

Cobertura del diseño asistido por computadora

La mayor parte de los proyectos de diseño de circuitos digitales modernos requiere el uso de métodos y herramientas de diseño asistido por computadora (CAD). Por esta razón, el CAD se trata, al final de todos los capítulos, lo que permite aplicar los métodos CAD a los conceptos y principios de diseño fundamentales presentados en cada capítulo.

La cobertura de los métodos y herramientas de CAD se diseñó de modo que fuera genérico, y no específico para las herramientas de un fabricante en particular. Esto permitirá a los estudiantes aplicar los conceptos a las herramientas CAD de que dispongan, incluidos los paquetes muy completos que se ejecutan en estaciones de trabajo de ingeniería de empresas como Mentor Graphics, Cadence y Viewlogic, y herramientas de menor nivel diseñadas para computadoras personales. Varias de estas últimas herramientas están disponibles a precio de lista para estudiantes y profesores.

La cobertura de CAD en los capítulos es la siguiente:

El capítulo 2 presenta el proceso de diseño asistido por computadora utilizado en el diseño y análisis de los circuitos lógicos y sistemas digitales. Los temas cubiertos incluyen la representación del diseño mediante diagramas esquemáticos y lenguajes de descripción de hardware, la captura de esquemas y la simulación lógica para la verificación del diseño y el análisis de tiempos.

El capítulo 3 analiza los métodos de CAD para la simplificación y organización de los circuitos lógicos combinatorios. El capítulo 4 extiende la cobertura de CAD al apoyo de los diseños jerárquicos y modulares de circuitos lógicos combinatorios. El capítulo 5 describe las herramientas de CAD para el diseño y modelado de circuitos por implantar en dispositivos lógicos programables, incluidos los lenguajes de descripción de hardware.

En la sección sobre circuitos secuenciales, el capítulo 8 analiza los métodos de CAD utilizados para el diseño y análisis de circuitos lógicos secuenciales, incluido el análisis de tiempos y la detección de las violaciones a las restricciones de tiempo. El capítulo 11 amplía este análisis a los métodos empleados para modelar circuitos lógicos secuenciales que se implantarán en dispositivos lógicos programables.

Apoyo a laboratorios

Los cursos de diseño digital utilizan con frecuencia experimentos de laboratorio para reforzar los conceptos presentados en clase. En algunos casos, se utiliza la captura de esquemas u otros métodos de CAD para modelar circuitos con diversos grados de complejidad, así como las herramientas de simulación para el estudio del funcionamiento de estos circuitos. Este texto apoya ambos tipos de laboratorio, basados en CAD o en tarjetas tradicionales.

El laboratorio con tarjetas tradicionales implica por lo general la construcción de circuitos digitales con módulos de circuitos TTL estándar integrados a pequeña escala (SSI) y a mediana escala (MSI). En todo el libro se analizan muchos ejemplos de estos módulos, comentando el diseño y la operación de cada módulo y el diseño de circuitos de nivel superior a partir de estos módulos.

Además de breves ejercicios de laboratorio, con frecuencia es recomendable utilizar proyectos de diseño amplios para que los estudiantes asimilen los diversos conceptos aprendidos en un curso. A fin de ilustrar los pasos de planeación y diseño de tales proyectos, el capítulo final del texto presenta cuatro casos basados en proyectos realizados por estudiantes en la North Carolina State University y la Auburn University.

Descripción de los capítulos

El material del presente texto está organizado en varias secciones. En cada sección, primero se desarrollan los conceptos y la teoría fundamentales con objeto de proporcionar una base sólida. Después, la teoría se aplica al diseño y análisis de circuitos sencillos, y se amplía al diseño de circuitos óptimos. Por último, se analizan los aspectos y métodos prácticos de diseño, incluido el uso de métodos de diseño modular, técnicas de diseño asistido por computadora, y dispositivos lógicos programables. En cada sección se presentan amplios ejemplos para ilustrar y reforzar los conceptos presentados en ella.

Antecedentes

Puesto que no hemos supuesto requisitos previos, los dos primeros capítulos presentan material básico que ayudará a comprender el diseño de circuitos digitales.

El **capítulo 0** presenta los circuitos y las computadoras digitales, incluidos los principales componentes de hardware y software de una computadora.

El **capítulo 1** presenta los sistemas numéricos y la representación de la información, con énfasis en los códigos binarios utilizados para representar los números y demás información en las computadoras digitales y otros circuitos. También se analiza la aritmética con números binarios, como preludio al diseño de los circuitos digitales de computadoras que realizan tales operaciones.

Circuitos lógicos combinatorios

El análisis y diseño de los circuitos lógicos combinatorios es el tema de la siguiente sección del libro, que inicia con ciertos fundamentos en el capítulo 2 y pasa por la optimización en el capítulo 3, el diseño modular en el capítulo 4 y el diseño con lógica programable en el capítulo 5.

El **capítulo 2** inicia con una presentación de las álgebras booleana y de conmutación, que forman la base del diseño de circuitos digitales. A continuación se presentan las compuertas lógicas digitales, seguidas por las técnicas de análisis de los circuitos construidos con compuertas básicas. A continuación se presentan la síntesis y el diseño de circuitos lógicos a partir de diversos tipos de especificaciones. El capítulo concluye con una introducción al diseño de circuitos lógicos digitales asistido por computadora.

El **capítulo 3** presenta algoritmos y métodos para simplificar los circuitos lógicos combinatorios. Se presentan con detalle los mapas de Karnaugh y el

método tabular de Quine-McCluskey, para después analizar los métodos asistidos por computadora para simplificar los circuitos lógicos combinatorios.

El **capítulo 4** analiza el diseño modular jerárquico de los circuitos digitales. Se describe el diseño y empleo de diversos módulos en tales diseños, incluidos los decodificadores, multiplexores y circuitos aritméticos. Como conclusión del análisis se presenta el apoyo con herramientas de CAD de las actividades del diseño modular jerárquico.

El **capítulo 5** describe el funcionamiento básico de los dispositivos lógicos programables y la implantación de los circuitos lógicos combinatorios con arreglos programables. Se describen las tres arquitecturas básicas de dispositivos, PLA, PROM y PAL, junto con ejemplos de módulos disponibles en el mercado. Se presentan las herramientas de CAD que apoyan el modelado de circuitos lógicos combinatorios que se implantarán mediante dispositivos programables.

Circuitos lógicos secuenciales

Los circuitos lógicos secuenciales, que implican el uso de memoria, se analizan en la siguiente sección del libro. El capítulo 6 describe los elementos de memoria utilizados en los circuitos secuenciales y el capítulo 7 examina el diseño y funcionamiento de varios módulos estándar de circuitos basados en estos elementos de memoria. El capítulo 8 presenta los fundamentos del análisis y diseño de circuitos síncronos, mientras que el capítulo 9 analiza los métodos para optimizar tales circuitos. El capítulo 10 analiza los problemas particulares asociados al análisis y diseño de circuitos secuenciales asíncronos. Por último, el capítulo 11 describe el uso de los dispositivos lógicos programables en el diseño de un circuito secuencial.

El **capítulo 6** comienza presentando los circuitos lógicos secuenciales, destacando el papel que desempeñan los elementos de memoria en tales circuitos. A continuación se analiza el diseño y funcionamiento de los dos tipos básicos de dispositivos de memoria, las *latches* y los *flip-flops*, a la vez que se describen las características de varios módulos disponibles en el mercado que contienen tales dispositivos.

El **capítulo 7** describe el diseño y funcionamiento de varios módulos de circuitos lógicos secuenciales estándar, como los registros, los registros de corrimiento y los contadores. Para cada tipo de módulo, se presenta el diseño y la teoría de operación básicos para después describir las características y empleo de varios módulos TTL estándar representativos.

El **capítulo 8** presenta los fundamentos y las técnicas para el análisis y la síntesis de los circuitos lógicos secuenciales asíncronos, incluidos los diagramas de tiempos, las tablas de estados y las tablas de excitación de los *flip-flops*. El capítulo concluye con un panorama de los métodos de CAD para modelar y simular la operación de los circuitos secuenciales síncronos y analizar las características particulares de tales circuitos con respecto del tiempo.

El **capítulo 9** analiza la optimización de los circuitos lógicos secuenciales síncronos. Se presentan métodos para eliminar los estados redundantes y así reducir el número de elementos de memoria necesarios para implantar un diseño, y los métodos para la asignación óptima de las variables de estado con el fin de minimizar la cantidad de compuertas lógicas combinatorias necesarias.

El **capítulo 10** analiza los circuitos secuenciales asíncronos de modo pulso y de modo fundamental. Se presentan métodos para el análisis y la síntesis de cada tipo de circuito, incluidos la identificación de competencias en los circuitos de modo fundamental y algunos métodos para evitar las competencias críticas.

El **capítulo 11** concluye la sección sobre circuitos secuenciales con una descripción de los dispositivos lógicos programables utilizados para implantar circuitos secuenciales síncronos y asíncronos, incluidos los PAL y PLA registrados y los dispositivos flexibles basados en macroceldas. También se tratan los arreglos de compuertas programables mediante campos. El capítulo incluye un panorama sobre los métodos de CAD para el modelado de circuitos secuenciales que serán sintetizados con dispositivos lógicos programables.

Verificación y diseño para la verificabilidad

El **capítulo 12** proporciona una introducción a las fallas en los circuitos lógicos digitales y los métodos de verificación, incluido el proceso de obtención de conjuntos de prueba para los circuitos lógicos. La verificación de un circuito digital representa un costo importante, sobre todo cuando los circuitos son muy grandes. El diseño para la verificabilidad es crítico para facilitar la verificación y minimizar el costo de la misma. Por tanto, este capítulo analiza varias técnicas de diseño de circuitos digitales que pueden mejorar la verificabilidad en los niveles de compuertas y tarjetas de circuitos, incluido el uso de circuitos de verificación integrados.

Estudios de casos de diseño digital

El **capítulo 13** concluye el texto presentando cuatro estudios de caso basados en proyectos reales y completos de diseño digital realizados por estudiantes de la North Carolina State University y la Auburn University: una máquina tragamonedas, un sistema para abrir un automóvil sin llave, un controlador de tránsito para coordinar el tráfico en ambos sentidos en un camino de un solo carril, y un controlador para caja registradora.

Sugerencias para esquemas de cursos

El material de este libro se puede utilizar en un curso trimestral o semestral, o podría extenderse a dos trimestres. Un curso trimestral (10 semanas) podría emplear el siguiente esquema.

Capítulo 0: Introducción general
Capítulo 1: Códigos de números binarios y aritmética binaria
Capítulo 2: Álgebra booleana y funciones de conmutación, compuertas lógicas, análisis y diseño de circuitos combinatorios
Capítulo 3: Minimización: un método (por lo general, mapas K)
Capítulo 4: Diseño modular jerárquico y módulos de circuitos estándar
Capítulo 6: Funcionamiento y diseño básico de los *flip-flops* y *latches*

Capítulo 7: Módulos sencillos para un registro de corrimiento secuencial y un contador
Capítulo 8: Análisis y síntesis de los circuitos secuenciales síncronos

Un segundo curso de 10 semanas podría invertir más tiempo en el diseño asistido por computadora, la lógica programable, los circuitos asíncronos y la verificación.

Un curso semestral de 16 semanas podría simplemente seguir el esquema del libro, agregando los temas de optimización de los capítulos 3 y 9, el tratamiento de los dispositivos lógicos programables de los capítulos 5 y 11 y la verificación del capítulo 12.

Agradecimientos

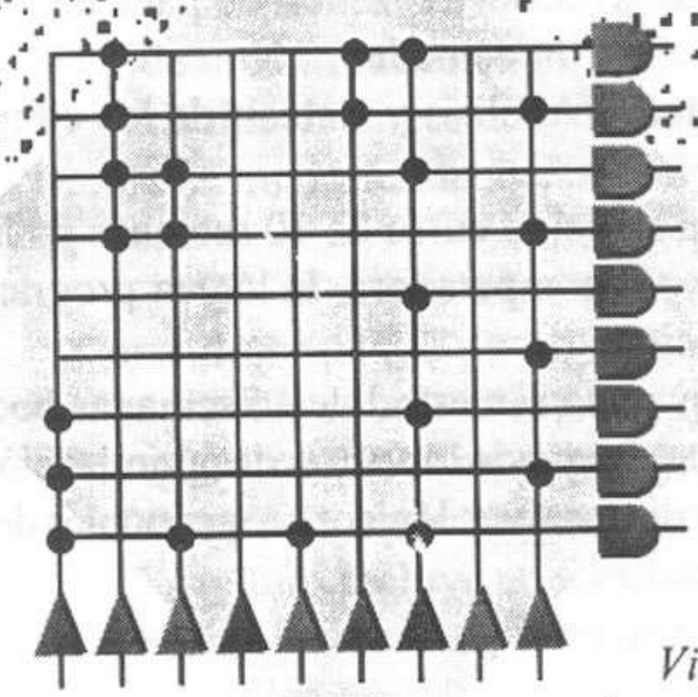
Los autores aprecian la colaboración de los estudiantes de la Auburn University, la North Carolina State University y la University of Texas at Arlington, quienes usaron el manuscrito en clase en lugar de un texto terminado, así como la de los colegas y asistentes graduados que participaron en la impartición de cursos derivados del manuscrito y ofrecieron muchas sugerencias valiosas. Entre ellos se encuentran Bruce Tucker, Bill Dillard, el profesor Adit Singh, el profesor Dharma Agrawal, la profesora Alexandra Duel-Hallen, Kam Yee, el profesor Hee Yong Youn y el profesor Vijay K. Raj.

Los autores también queremos agradecer a nuestro editor Don Fowley sus múltiples y útiles sugerencias y contribuciones durante la preparación del manuscrito y el diseño del libro. Agradecemos en particular a Meredith Nelson su apoyo en la transcripción de la primera versión y a Gregory Nelson su trabajo en el Manual de soluciones.

Los comentarios de varios revisores fueron una valiosa ayuda para desarrollar el manuscrito, en particular los de Michael A. Driscoll de Portland State University, David Bray de Clarkson University, Karan Watson de Texas A & M, Dong Ha de Virginia Polytechnic Institute y Kewal Saluja de la University of Wisconsin-Madison.

Por último, queremos expresar nuestro aprecio a nuestras esposas, Margaret, Susan, Marsha y Edie, por su apoyo y paciencia durante el proceso, en apariencia interminable, de desarrollo de este manuscrito.

Victor P. Nelson
H. Troy Nagle
Bill D. Carroll
J. David Irwin



Vivimos en una era que los sociólogos han llamado la revolución de las computadoras. Como cualquier otra revolución, tiene una amplia difusión, es penetrante y tendrá un efecto duradero en la sociedad. Es tan fundamental para la economía y el orden social actuales como lo fue la Revolución Industrial en el siglo XIX. Afectará los patrones de pensamiento y el estilo de vida de cada individuo. Mientras que el principal efecto de la Revolución Industrial fue el aumento de nuestras posibilidades físicas, la revolución de las computadoras amplía nuestro poder mental.

Las computadoras están formadas por elementos electrónicos, mecánicos y ópticos, conocidos como hardware, así como por programas y datos conocidos como software. Este libro presenta el tema del hardware de las computadoras. En particular, estudiaremos el análisis y diseño de los circuitos lógicos que forman la base de la mayor parte del hardware electrónico de las computadoras, pero primero repasaremos la historia y la organización de la computadora digital.

Introducción

0.1 Historia de la computación

Una *computadora* es un dispositivo capaz de resolver problemas o controlar información, según una secuencia predeterminada de instrucciones (o *programa*), mediante algún proceso mecánico o eléctrico. Desde que las personas comenzaron a resolver problemas hace miles de años, se han buscado diversas formas de simplificar las tareas de resolución de problemas. La automatización de las operaciones aritméticas ha sido un interés primordial durante estos milenios. El surgimiento de la tecnología de las computadoras proporcionó una forma económica de realizar aritmética sencilla, y, conforme maduró la tecnología, las técnicas de computación se ampliaron con rapidez para resolver problemas numéricos complejos, guardar, recuperar y comunicar información, controlar robots, aparatos, automóviles, juegos, fábricas y una amplia gama de procesos y máquinas. Lo más interesante es que esta revolución de las computadoras ¡ha sucedido dentro de los últimos 50 años! A continuación presentamos un resumen de estos avances.

0.1.1 Los inicios: computadoras mecánicas

Tal vez la primera computadora fue el ábaco, utilizado en el Oriente durante más de 3000 años. Este dispositivo, aún en uso, tuvo poca competencia hasta cerca de 1600, cuando John Napier utilizó los logaritmos como fundamento para un dispositivo que multiplicaba números. Su trabajo condujo a la invención de la regla de cálculo. Después, en 1642, Blaise Pascal construyó una máquina sumadora que tenía engranes parecidos al odómetro moderno.

En 1820, Charles Babbage construyó el primer dispositivo que utilizaba los principios de las computadoras modernas. Su invento, la *máquina de diferencias*, evaluaba polinomios mediante el método de diferencias finitas (ver [1]). También imaginó una máquina mecánica semejante a las computadoras modernas, con una unidad de almacenamiento y aritmética. Sin embargo, la precisión requerida para construir los engranes mecánicos estaba mas allá de la capacidad de los artesanos de su tiempo.

0.1.2 Primeras computadoras electrónicas

El primer progreso real hacia las computadoras digitales electrónicas ocurrió a fines de la década de 1930, cuando Howard Aiken de la Universidad de Harvard y George Slibitz de Bell Telephone Laboratories desarrollaron una calculadora automática utilizando redes de relevadores; el *relevador* es un interruptor controlado en forma electromagnética. Durante la Segunda Guerra Mundial se desarrollaron otras máquinas con relevadores para los cálculos de balística en artillería. Aunque estas máquinas eran relativamente lentas y comparativamente grandes, demostraron la versatilidad de la computadora electrónica. Después, a principios de la década de 1940, John Mauchly y J. Presper Eckert, Jr., de la Universidad de Pennsylvania, diseñaron y construyeron una computadora con bulbos, que llamaron Integrador y Calculador Numérico Electrónico (ENIAC); fue terminada en 1945 y se instaló en un campo de pruebas en Aberdeen, Maryland. ENIAC utilizaba 18 000 bulbos y requería una cantidad de energía tremenda; su probabilidad de fallas era alta y resultaba difícil programarla, ya que se necesitaba un tablero de conexiones manuales.

Entonces surgieron tres importantes descubrimientos, que iniciaron la rápida evolución hacia la computadora digital actual. Primero, John von Neumann propuso que el programa residiera en la memoria de la computadora, donde se podría modificar a voluntad, lo que resolvió las dificultades de programación de la ENIAC; segundo, John Bardeen, Walter H. Brattain y William Shockley inventaron el transistor en 1947, lo que redujo en forma drástica las necesidades de espacio y energía al reemplazar al bulbo; y tercero, J. W. Forrester y su grupo en el Instituto Tecnológico de Massachusetts desarrollaron la memoria de núcleo magnético, que permitió aumentar la capacidad de almacenamiento considerablemente.

0.1.3 Las primeras cuatro generaciones de computadoras

ENIAC y las demás computadoras de bulbos que aparecieron a fines de la década de 1940 y durante la década de 1950 han sido llamadas la *primera generación* de computadoras digitales. El surgimiento de los transistores a fines de la década de 1950 trajo consigo la *segunda generación* de máquinas, con un tamaño menor, más rápidas, y con mucho mayor capacidad que sus antecesoras. A fines de la década de 1960 y durante la década de 1970 apareció la *tercera generación* de máquinas. Éstas se caracterizaron por utilizar *circuitos integrados,* que constan de paquetes miniatura con varios circuitos de transistores, lo que llevó a una reducción de tamaño aún más drástica. Las mejoras en las tecnologías de empaque y de memoria también contribuyeron a perfeccionar las máquinas de la tercera generación.

A fines de la década de 1960 surgió la minicomputadora. Además de las grandes máquinas complejas, con frecuencia llamadas *mainframes*, muchos fabricantes ofrecieron estas computadoras más pequeñas, con una capacidad limitada y de propósito general. Las minicomputadoras, que recibieron tal nombre por su tamaño y costo, se utilizaron en muchas aplicaciones diversas y han desempeñado un papel fundamental en la popularización del uso de la computadora. La minicomputadora generalizó el uso de las computadoras en las comunidades científica y de ingeniería, y las máquinas se abrieron camino en los laboratorios de investigación de las industrias y las universidades. El control computarizado de procesos en la industria se hizo cosa común.

La *cuarta generación* de computadoras se introdujo a fines de la década de 1970 y principios de la década de 1980 con la aparición de las máquinas basadas en componentes de hardware con circuitos de integración a gran escala (LSI) y a muy grande escala (VLSI). Con la VLSI se pudieron construir computadoras pequeñas pero potentes, conocidas como computadoras personales o estaciones de trabajo. El componente central de estas máquinas es el *microprocesador*, que es toda una unidad central de procesamiento de una computadora implantada en un componente VLSI. Intel Corporation y Motorola han ido a la vanguardia del desarrollo de la tecnología de microprocesadores. Este avance se ilustra en la figura 0.1, la cual muestra la evolución en un periodo de 20 años de los circuitos microprocesadores VLSI de Intel utilizados en las computadoras personales IBM y compatibles con IBM.

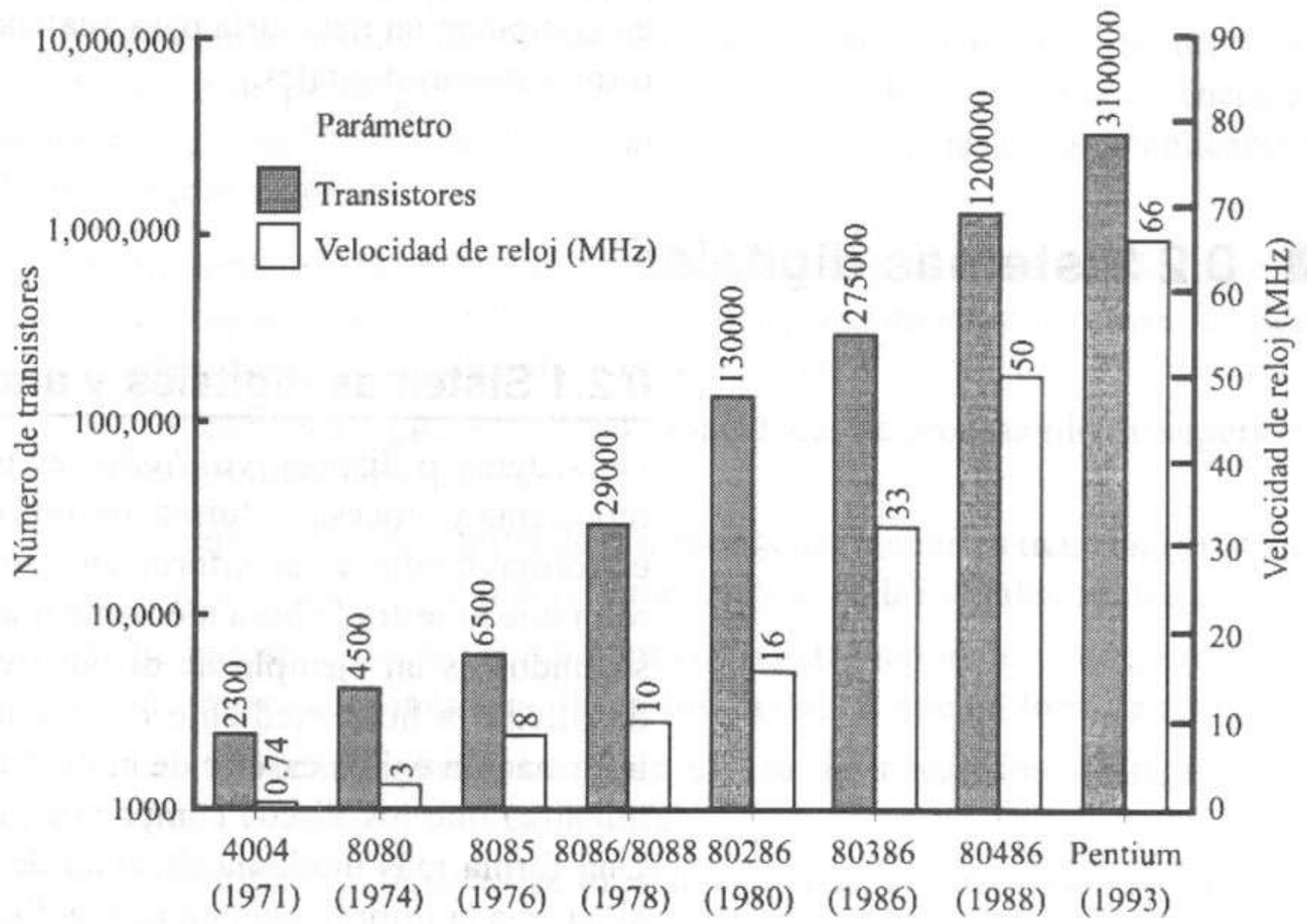


Figura 0.1 Evolución de los microprocesadores Intel.

Es posible que la aparición de computadoras personales como la PC de IBM, basada en microprocesadores Intel, y la Apple Macintosh, basada en los microprocesadores Motorola, haya tenido el efecto más drástico para ampliar la gama de aplicaciones de las computadoras que cualquier otro hecho. Antes de la amplia difusión de la computadora personal, se podía afirmar que la mayor parte de las computadoras era utilizada por expertos en computación. Actualmente, las computadoras son utilizadas por igual y en forma común por expertos y no expertos. También las redes de computadoras han proliferado durante la cuarta generación. Las redes han incrementado el acceso a las computadoras y han generado nuevas aplicaciones, como el correo electrónico.

0.1.4 La quinta generación y más allá

¿Cuándo comenzará la quinta generación de computadoras? ¿Ha comenzado ya? Si utilizamos la medida clásica, el cambio a una nueva base de tecnología de hardware, la respuesta es no. ¿Debería ser la tecnología de hardware el único indicador de las generaciones de computadoras? Tal vez no. Es claro que los avances en el software han tenido un efecto profundo en la forma en que son utilizadas las computadoras. Las nuevas interfaces de usuario, como la activación por voz, o los nuevos paradigmas computacionales, como el procesamiento en paralelo y las redes neuronales, también podrían caracterizar a la siguiente generación de máquinas. Como sea, es probable que el procesamiento en paralelo, la inteligencia artificial, el procesamiento óptico, la programación visual y las redes de gigabits desempeñarán un papel clave en los sistemas de cómputo del futuro. Es probable que estemos usando la quinta generación de computadoras durante algún tiempo antes de que eso sea evidente.

Con esta perspectiva, repasaremos ahora cierta terminología importante en computación necesaria para analizar y diseñar circuitos de computadoras y otros sistemas digitales.

0.2 Sistemas digitales

0.2.1 Sistemas digitales y analógicos

Un sistema o dispositivo *digital* es un sistema en el que la información se representa y procesa en forma discreta en vez de continua. Los sistemas basados en formas continuas de información son sistemas o dispositivos *analógicos*. Un reloj que muestra la hora mediante manecillas para las horas, los minutos y los segundos es un ejemplo de dispositivo analógico, mientras que un reloj que despliega la hora mediante dígitos decimales es un dispositivo digital. La información en los casetes de audio tradicionales se graba en forma analógica, mientras que los discos compactos guardan la información en forma digital. Una forma más moderna de cinta de audio, la cinta de audio digital (DAT), almacena la información en forma digital.

Por ejemplo, la figura 0.2a muestra una señal analógica que podría aparecer en un fragmento de cinta magnética de audio. La figura 0.2b presenta la misma señal muestreada a intervalos de tiempo uniformes y convertida en un número discreto de valores. La figura 0.2c muestra esta información en forma digital, donde cada muestra se representa con un número binario escrito en forma vertical en la cinta.

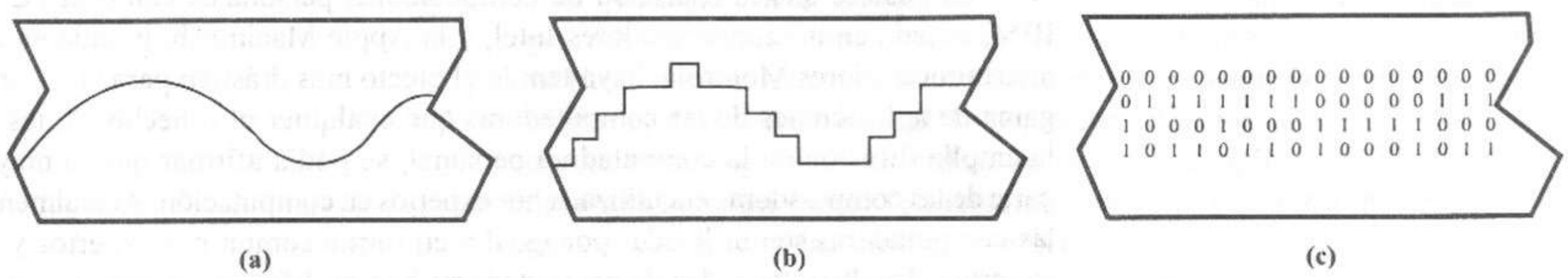


Figura 0.2 Cinta magnética que contiene formas analógicas y digitales de una señal. **(a)** Forma analógica. **(b)** Forma analógica muestreada. **(c)** Forma digital.

Aunque la computadora moderna es el ejemplo más visible de sistema digital, existen muchos otros ejemplos, que incluyen a los relojes digitales, los controladores de semáforos y las calculadoras de bolsillo. Todos estos ejemplos (distintos de la computadora) son sistemas con funciones fijas que no pueden ser modificadas por el usuario. En contraste, la computadora es un sistema *programable*; es decir, se puede modificar para cambiar las tareas o aplicaciones que realiza. En otras palabras, las computadoras son sistemas de propósito general, mientras que los otros ejemplos tienen una aplicación específica.

En una tendencia de rápido aumento, las computadoras se utilizan en lugar de los circuitos de aplicación específica en productos tales como motores de automóviles, aparatos caseros y juegos electrónicos, mediante el desarrollo de programas que realizan las tareas necesarias de la aplicación y la inserción posterior de la computadora programada en el producto. La capacidad para programar una computadora de modo que realice cualquier tarea permite utilizar las computadoras insertadas en lugar de una amplia variedad de circuitos fijos, usualmente con un costo mucho menor.

Las computadoras analógicas y otros sistemas analógicos se utilizaron mucho antes de perfeccionar los dispositivos digitales. ¿Por qué entonces los sistemas digitales han sustituido a los analógicos en muchas áreas de aplicación? Existen varias razones.

- En general, las técnicas digitales ofrecen más flexibilidad que las analógicas, en el sentido de que pueden programarse con mayor facilidad para realizar cualquier algoritmo deseado.
- Los circuitos digitales ofrecen capacidades de procesamiento superiores en términos de velocidad.
- La representación digital de la información numérica tiene una precisión y un alcance de rango mayor que el caso de las señales analógicas.
- La implantación de las funciones de almacenamiento y recuperación de información es más sencilla en forma digital que en forma analógica.
- Las técnicas digitales permiten el uso de mecanismos integrados de detección y corrección de errores.
- Los sistemas digitales se prestan mejor a la miniaturización que los analógicos.

0.2.2 Jerarquía de diseño de un sistema digital

Los sistemas digitales pueden diseñarse y estudiarse en muchos niveles de abstracción diferentes, que van desde un modelo puramente de comportamiento, en el que no se especifican los detalles del hardware, hasta un nivel físico, en el que sólo se especifican las estructuras de los materiales físicos. La tabla 0.1 enumera diversos niveles de abstracción de diseño.

Los niveles de sistema y de registros

En su máximo nivel, un sistema digital se puede ver como uno o más módulos funcionales con cierta interacción. El comportamiento de cada módulo se describe sin especificar los detalles de la implantación. Por ejemplo, una computadora de escritorio vista en el nivel de sistema incluye un microprocesador, módulos de memoria y circuitos de control para el monitor, el teclado, la impresora y otros dispositivos periféricos.

TABLA 0.1 JERARQUÍA DE ABSTRACCIÓN DEL DISEÑO DE UN SISTEMA DIGITAL

Nivel de diseño	Nivel de abstracción	Grado de detalle	Tipo de modelo
Sistema	Máximo	Mínimo	De comportamiento
Registro	"	"	De comportamiento / estructural
Compuerta	"	"	Estructural
Transistor	"	"	Estructural
Físico	Mínimo	Máximo	Estructural

En el nivel de registros, un sistema digital se ve como un conjunto de elementos llamados *registros* que guardan información, interconectados de algún modo mediante líneas de señal. La información es procesada por el sistema al transferirla entre los registros a través de estas líneas de señal. En algunos casos, la información se transforma durante estas transferencias entre registros haciéndola pasar por uno o más módulos funcionales. Las figuras 0.3a y b muestran los modelos de nivel de sistema y de registros de un sistema digital

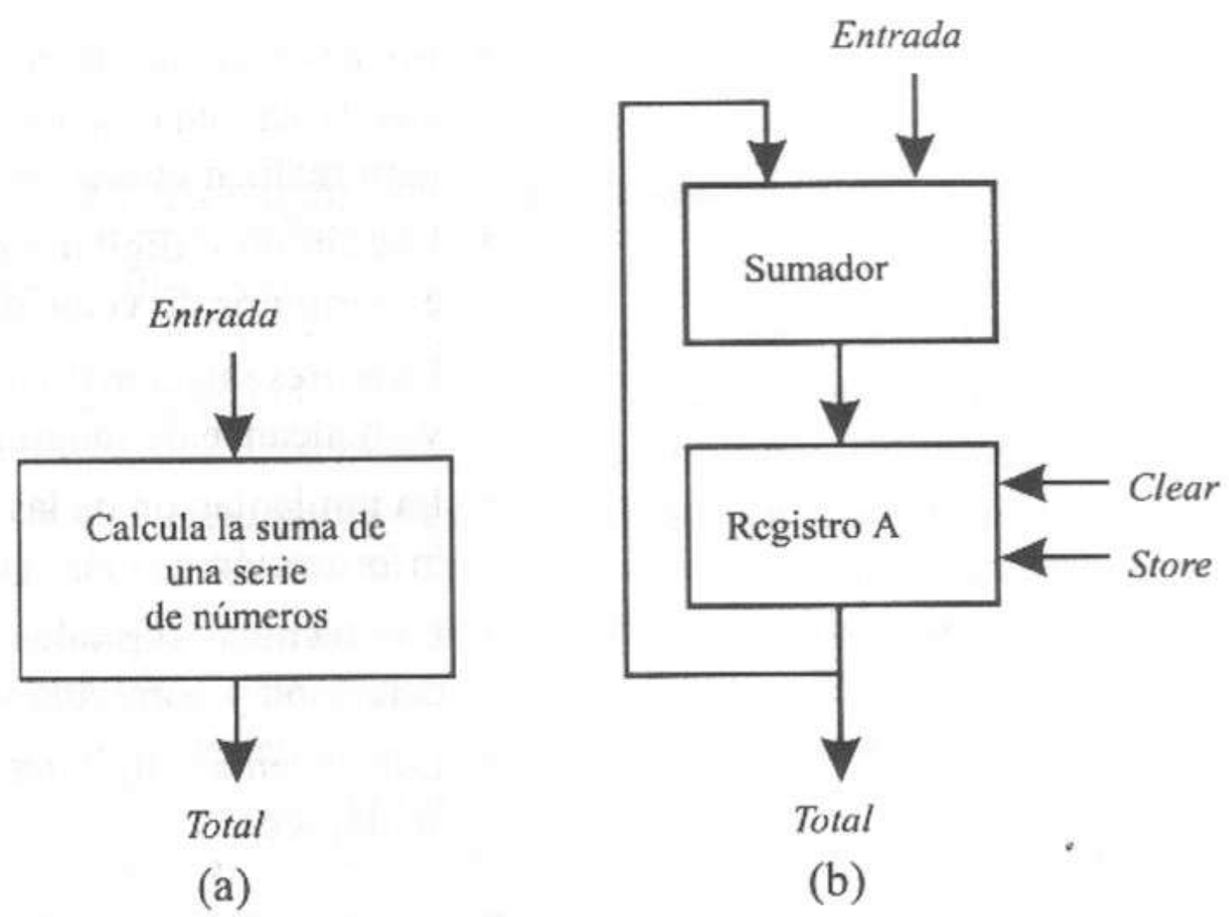


Figura 0.3 Modelos de un sistema digital que suma listas de números. **(a)** Nivel de sistema. **(b)** Nivel de registros.

que calcula la suma de una serie de números binarios, los cuales se proporcionan uno a la vez como datos del sistema. En el nivel de sistema, todo lo que conocemos es la función básica del sistema, que es calcular:

$$\text{Total} = \sum_{i=1}^{N} \text{Entrada}_i$$

En el nivel de registros, como en la figura 0.3b, se ve que el sistema incluye un registro de almacenamiento, *A*, y un circuito sumador. El *Total* se calcula limpiando primero el registro *A*, mediante la señal de control *Clear* y sumando después cada número $Entrada_i$ al contenido del registro *A*, reemplazando el contenido del *A* con la nueva suma mediante la señal de control *Store*. Por tanto,

la suma de una lista de números se calcula realizando las siguientes transferencias de registros en el orden correcto.

$$Clear: A \leftarrow 0$$

$$Store: A \leftarrow A + \text{entrada}$$

El nivel de compuertas

En su nivel más bajo, el comportamiento de un sistema digital se especifica como un conjunto de ecuaciones lógicas del álgebra de conmutación que puede realizarse en el hardware mediante circuitos lógicos. La unidad lógica mínima del hardware digital es la *compuerta*. Las compuertas son elementos de conmutación que implantan los operadores fundamentales del álgebra de conmutación. Las ecuaciones lógicas se materializan en hardware al interconectar compuertas que forman *circuitos lógicos combinatorios*, como se muestra en la figura 0.4. Observe que el circuito tiene seis compuertas. Las entradas en este ejemplo son $x_1,..., x_5$ y la salida $f(x_1,..., x_5)$ es una función que sólo depende del valor actual de las señales de entrada. Por tanto, una característica distintiva del circuito lógico combinatorio es que no tiene memoria de las señales de entrada previas. El análisis y diseño de los circuitos lógicos combinatorios ocupa una gran parte de este texto.

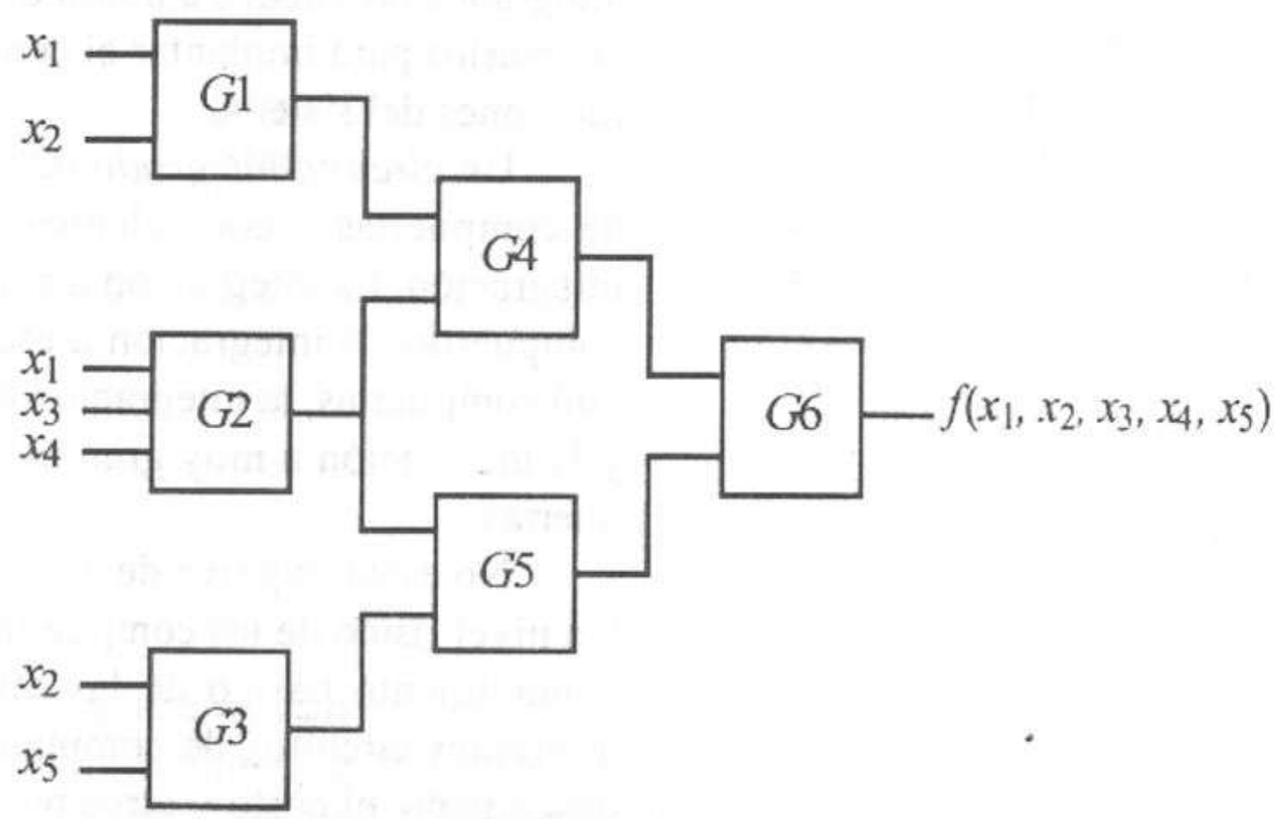


Figura 0.4 Un circuito lógico combinatorio con seis compuertas.

Todas las computadoras digitales tienen dispositivos de memoria llamados *registros* que sirven como espacios de almacenamiento temporal de la información. Estos registros y ciertas partes de la unidad de control se denominan circuitos lógicos secuenciales. Un circuito lógico *secuencial* es, en general, un circuito lógico combinatorio con memoria, como se modela en la figura 0.5. A diferencia de los circuitos lógicos combinatorios, las salidas de un circuito lógico secuencial son funciones que no sólo dependen del valor actual de las señales de entrada, sino también de la historia anterior de las entradas, reflejada por la información guardada en los registros. El análisis y diseño de los circuitos lógicos secuenciales es el segundo punto central del presente texto. Sólo cuando los lectores hayan dominado los fundamentos de los circuitos lógicos combinatorios y secuenciales podrán proceder con el diseño y construcción de hardware de sistemas digitales.

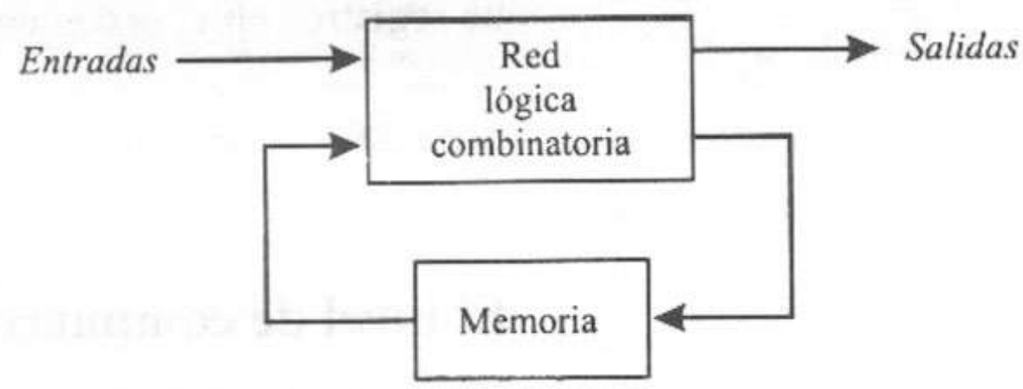


Figura 0.5 Circuito lógico secuencial.

Niveles de transistores y de diseño físico

Los circuitos lógicos combinatorios y secuenciales definen por completo el comportamiento lógico de un sistema digital. En última instancia, cada compuerta lógica se debe realizar con un circuito de transistores en un nivel más bajo, que a su vez se realizan combinando diversos semiconductores y otros materiales. Las tecnologías utilizadas para construir compuertas y otros elementos lógicos han evolucionado desde los dispositivos mecánicos, los relevadores, los bulbos y los transistores discretos hasta llegar a los circuitos integrados. La figura 0.6 ilustra varios de estos dispositivos. Las computadoras modernas y los sistemas digitales de aplicación específica se construyen por lo general mediante circuitos integrados ordenados a modo de realizar los registros y los circuitos de control necesarios para implantar el conjunto de instrucciones de la computadora o las funciones del sistema.

Un *circuito integrado* (CI) contiene varios elementos lógicos. El número de compuertas o equivalentes de compuertas por CI determina la escala de integración. La integración a pequeña escala (SSI) se refiere a los CI con 1 a 10 compuertas, la integración a escala media (MSI) corresponde a los CI de 10 a 100 compuertas, la integración a gran escala (LSI) de 100 a 10 000 compuertas, y la integración a muy gran escala (VLSI) a los CI con más de 100 000 compuertas.

No es un objetivo de este texto considerar el diseño a nivel de transistores y a nivel físico de las compuertas lógicas. Sin embargo, es importante tener un conocimiento básico de las diversas propiedades eléctricas y físicas de los diferentes circuitos de compuertas para poder evaluar la operación lógica, el desempeño, el costo y otros parámetros del diseño de un sistema digital.

Tecnologías electrónicas

Se han desarrollado numerosas *familias* de tecnologías electrónicas para proporcionar ciertas características, como velocidad, consumo de energía, densidad de empacado, funcionalidad y costo, preferidas por el diseñador del hardware. Por lo general, es imposible proporcionar todas las características deseadas en una familia. Por tanto, existe una búsqueda permanente de mejoras en las tecnologías ya probadas o de nuevas tecnologías. Las tablas 0.2 y 0.3 enumeran las tecnologías más importantes utilizadas desde los inicios de la era de los transistores y sus características correspondientes.

El empacado de compuertas lógicas y otros elementos lógicos ha cambiado de manera significativa con los años. Los primeros elementos lógicos electrónicos se construían por lo general con grandes bulbos, resistencias y condensadores discretos, montados en un chasis de aluminio, e interconectados con alambre de

TABLA 0.2 TECNOLOGÍAS ELECTRÓNICAS IMPORTANTES

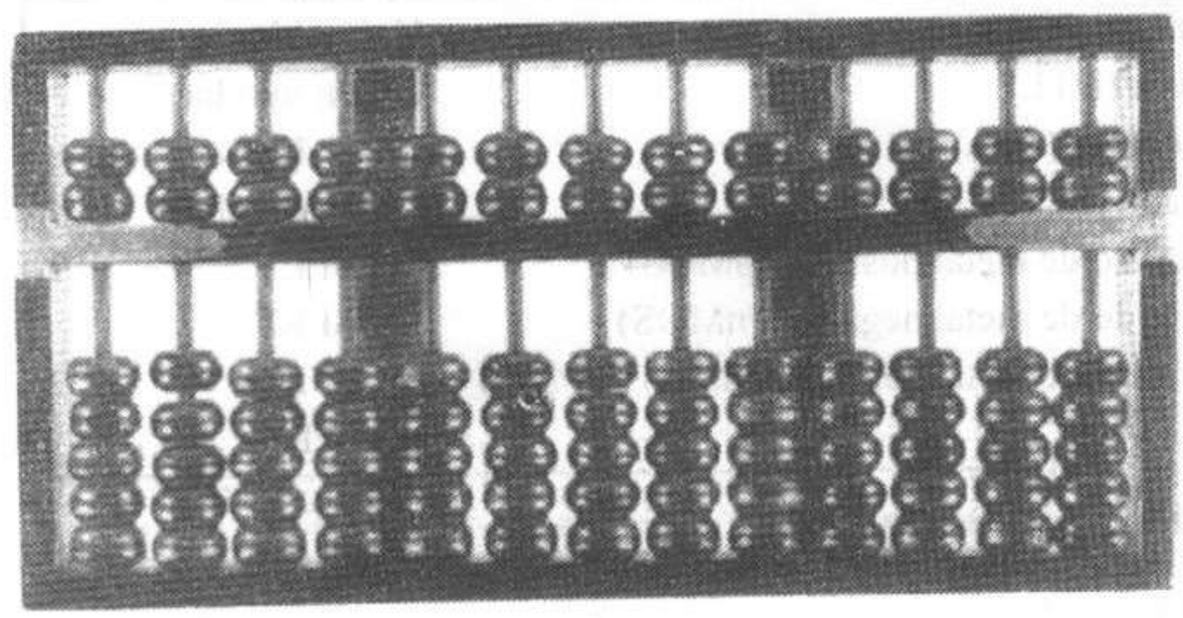
Tecnología	Tipo de dispositivo
Lógica resistor-transistor (RTL)	Unión bipolar
Lógica diodo-transistor (DTL)	Unión bipolar
Lógica transistor-transistor (TTL)	Unión bipolar
Lógica emisor-acoplado (ECL)	Unión bipolar
Semiconductor con óxido de metal positivo (pMOS)	MOSFET
Semiconductor con óxido de metal negativo (nMOS)	MOSFET
Semiconductor con óxido de metal complementario (CMOS)	MOSFET
Arseniuro de galio (GaAs)	MESFET

TABLA 0.3 CARACTERÍSTICAS DE LAS FAMILIAS DE TECNOLOGÍA ELECTRÓNICA

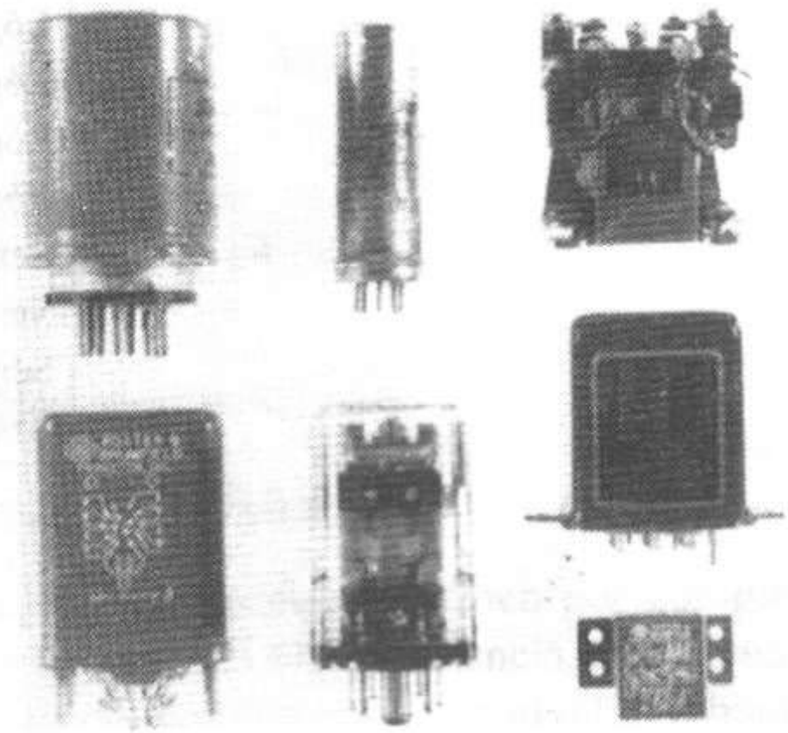
Tecnología	Consumo de energía	Velocidad	Empacado
RTL	Alto	Baja	Discreto
DTL	Alto	Baja	Discreto, SSI
TTL	Medio	Media	SSI, MSI
ECL	Alto	Alta	SSI, MSI, LSI
pMOS	Medio	Baja	MSI, LSI
nMOS	Medio	Media	MSI, LSI, VLSI
CMOS	Bajo	Media	SSI, MSI, LSI, VLSI
GaAs	Alto	Alta	SSI, MSI, LSI

cobre. Los avances en la tecnología de los bulbos redujeron el tamaño, y las tarjetas de circuitos impresos reemplazaron a los alambres de cobre. Más adelante, los transistores discretos reemplazaron a los bulbos, pero las resistencias, los condensadores y las tarjetas de circuitos impresos continuaron en uso, aunque sus tamaños fueron más reducidos. El surgimiento del circuito integrado a principios de la década de 1960 produjo una reducción adicional en el tamaño de las tarjetas de circuitos impresos y otros elementos pasivos.

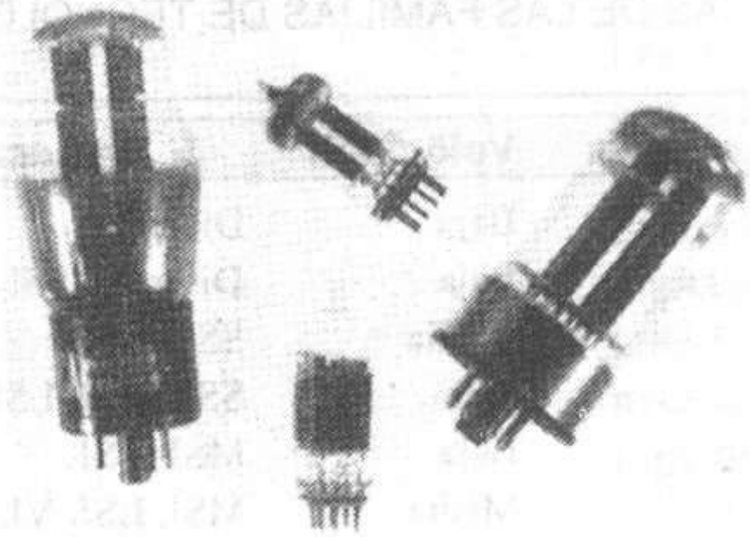
Los circuitos integrados pueden fabricarse en formas estándar, semiadaptada y adaptada. Los CI estándar proporcionan los componentes necesarios para construir sistemas para la mayor parte de las aplicaciones. Sin embargo, algunas aplicaciones podrían requerir circuitos semiadaptados o adaptados para cumplir con requisitos de funciones especiales, bajo costo o menor tamaño. Los circuitos adaptados se fabrican de acuerdo con las necesidades precisas de un cliente. Por otro lado, los circuitos semiadaptados se programan para satisfacer las necesidades de un cliente. El término *circuitos integrados específicos para una aplicación* (ASIC) se utiliza con frecuencia para describir los dispositivos semiadaptados.



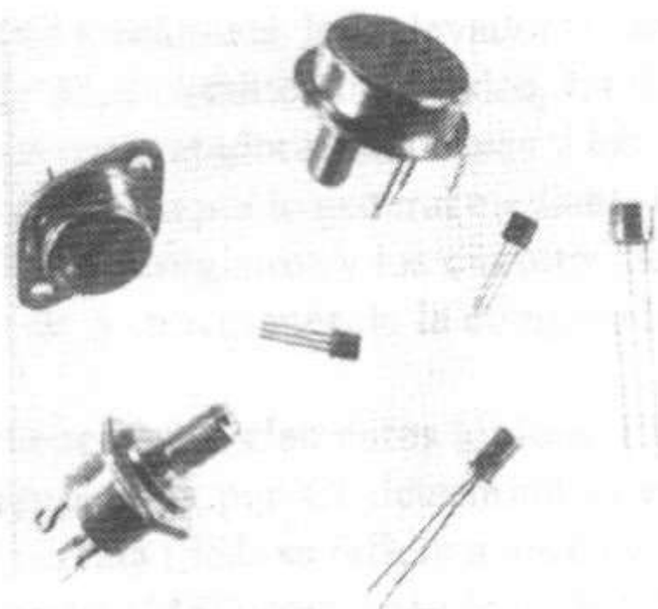
(a)



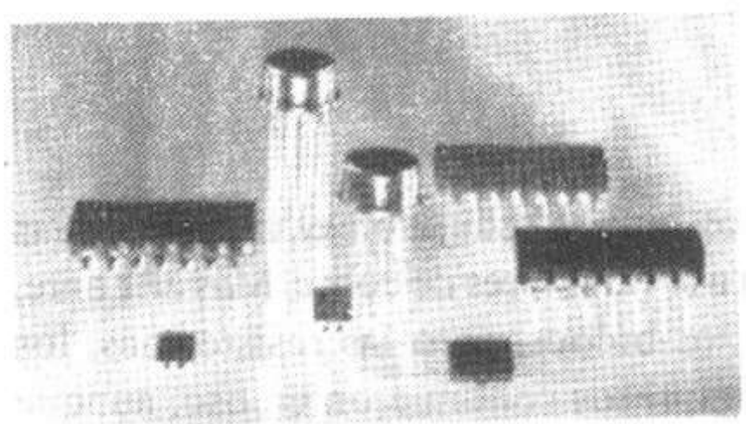
(b)



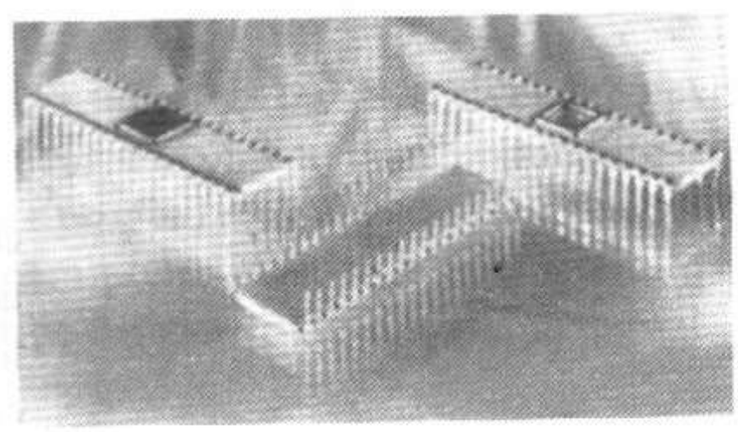
(c)



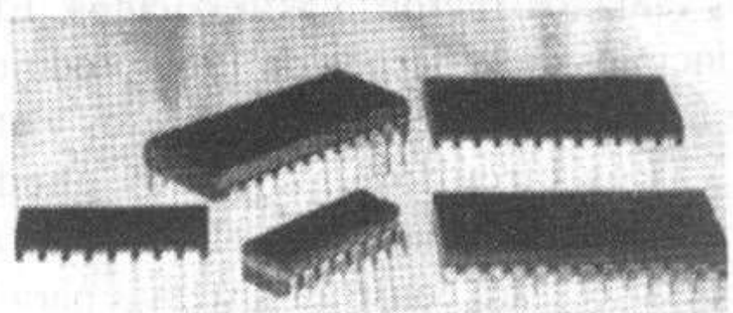
(d)



(e)

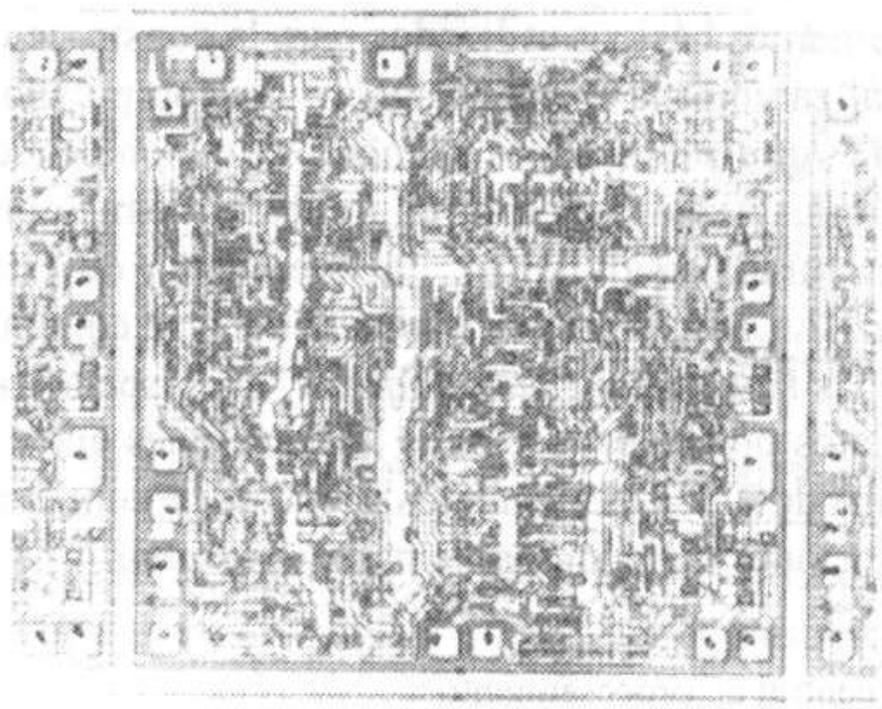


(g)

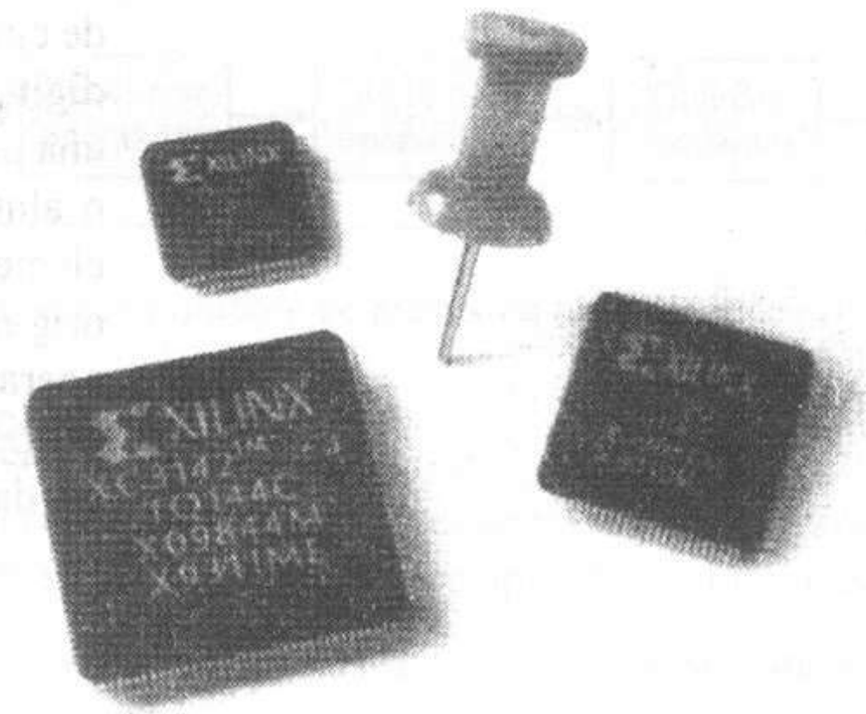


(f)

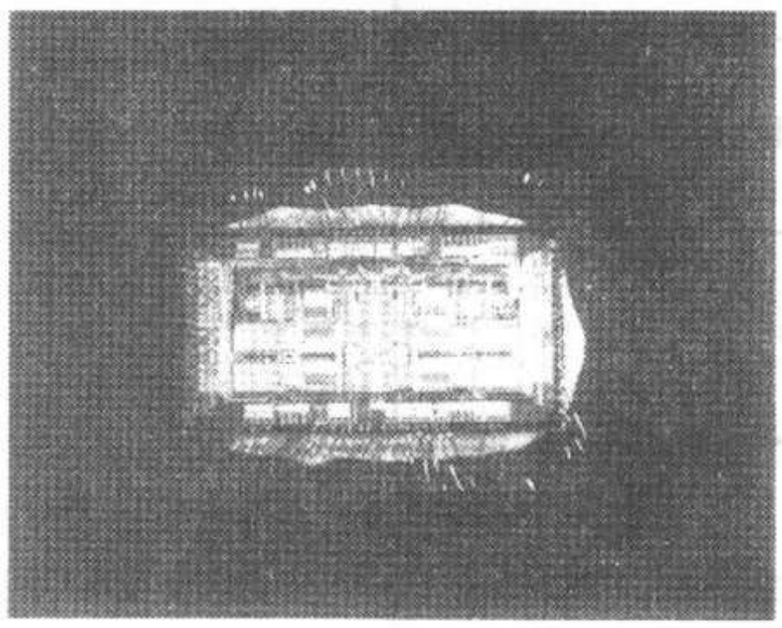
Figura 0.6 Fotografías de hardware de computadoras. **(a)** ábaco; **(b)** relevadores; **(c)** bulbos; **(d)** transistores; **(e)** circuitos integrados a pequeña escala; **(f)** circuitos integrados a mediana escala; **(g)** circuito integrado a gran escala.



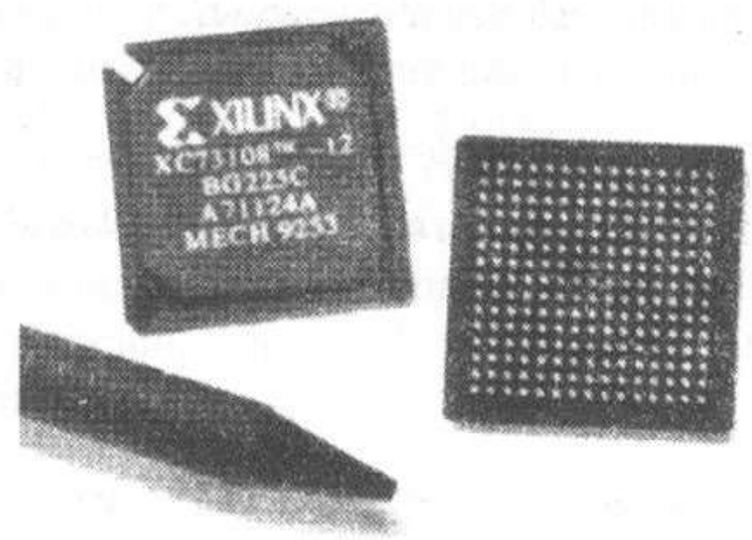
(h)



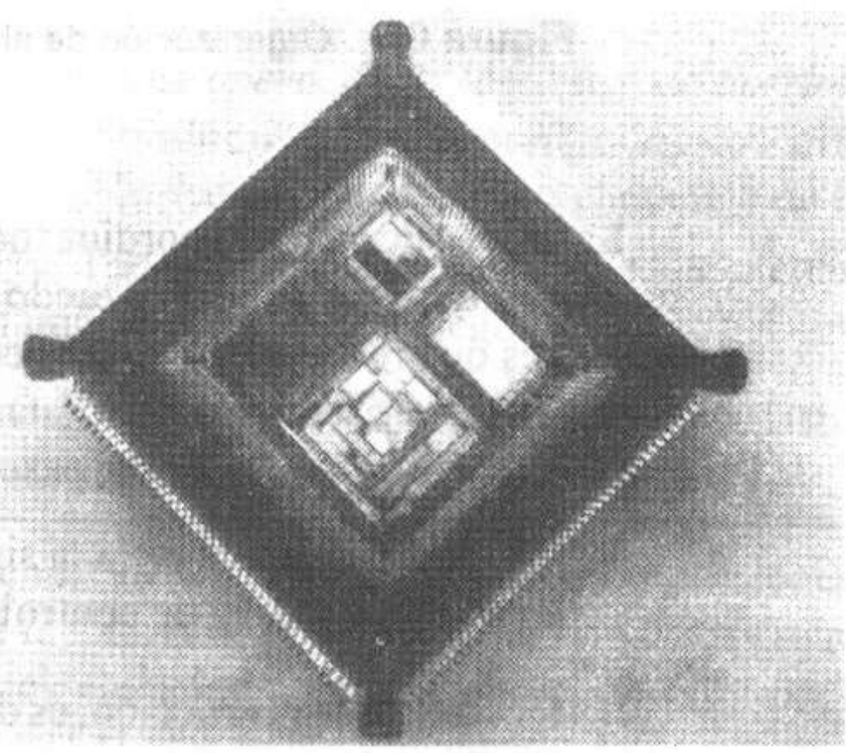
(i)



(j)



(k)



(l)

Figura 0.6 (Continuación). **(h)** vista interna de un *chip* de circuito integrado (MC74450) (cortesía de Motorola Semiconductor Products, Inc.); **(i)** vista externa de un dispositivo lógico programable en forma electrónica (cortesía de Xilinx); **(j)** vista interna de un dispositivo lógico programable electrónicamente (cortesía de Xilinx); **(k)** un circuito integrado a muy grande escala en un empaque plano con una retícula de pines para entrada/salida (cortesía de Xilinx); **(l)** módulo de varios *chips*, el cual contiene tres *chips*: un microprocesador, una memoria exclusiva para lectura programable en forma electrónica y un circuito integrado de aplicación específica (cortesía de Texas Instruments).

0.3 Organización de una computadora digital de programa almacenado

Ahora que ya conocemos los elementos básicos utilizados para la construcción de circuitos lógicos digitales, analizaremos la organización de una computadora digital. Una computadora digital es un sistema cuyos elementos funcionales son una unidad aritmética lógica (UAL), unidades de control, unidades de memoria o almacenamiento y equipo de entrada/salida (E/S). La interacción de estos elementos se muestra en la figura 0.7. Cada sistema de cómputo tiene un conjunto original de instrucciones, llamadas *instrucciones de máquina*, que especifican operaciones a realizar con los datos por la UAL y otras interacciones entre la UAL, la memoria y los dispositivos de E/S. Los elementos de memoria contienen los datos más una lista de instrucciones de máquina llamada *programa*.

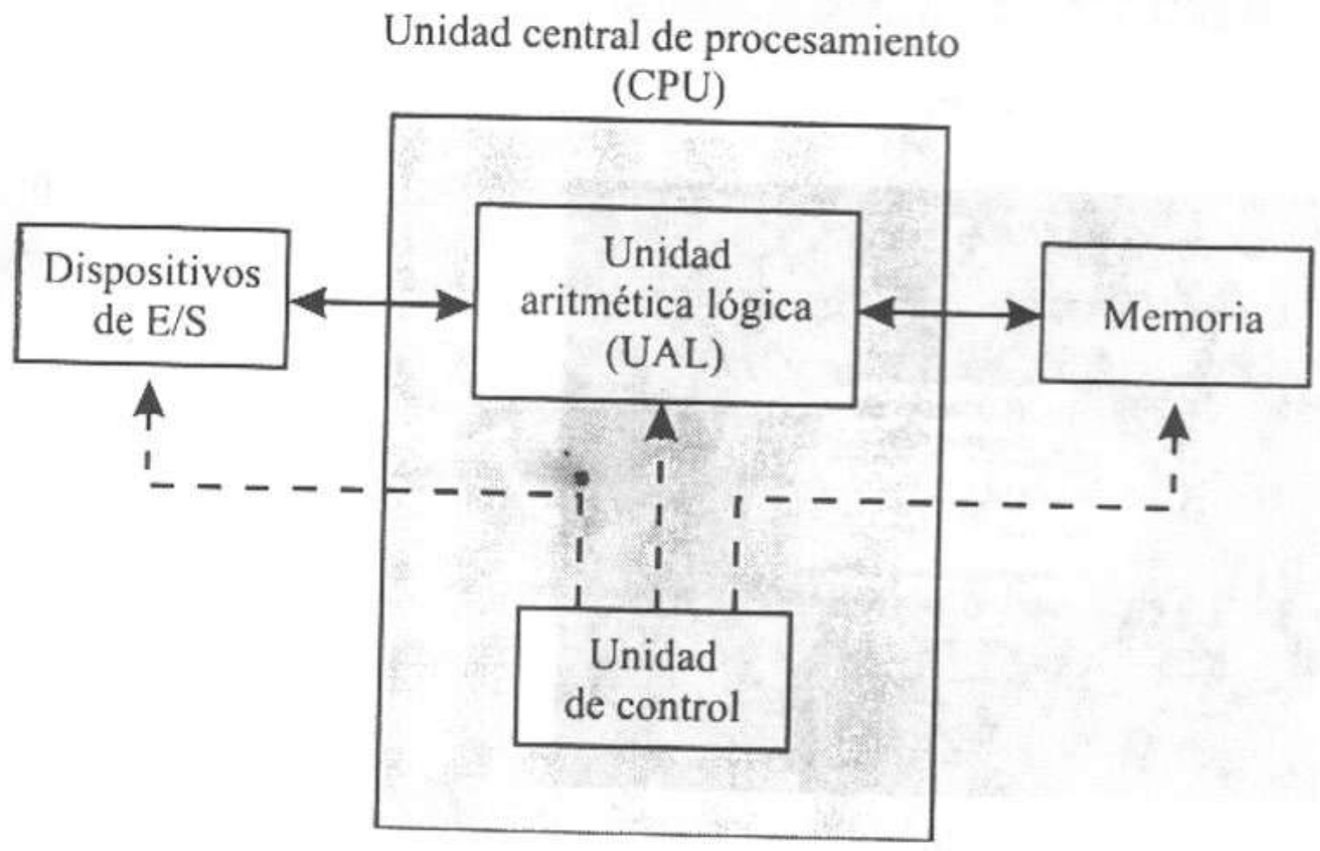


Figura 0.7 Organización de alto nivel de una computadora digital.

La unidad de control coordina todas las operaciones de la UAL, la memoria y los dispositivos de E/S, realizando un ciclaje continuo con un conjunto de operaciones que recuperan las instrucciones de la memoria para ejecutarlas. El *ciclo de instrucción* de una computadora digital sencilla, que se ilustra en la figura 0.8, incluye los siguientes pasos básicos:

1. Recuperar de la memoria la siguiente instrucción del programa actual y llevarla a la unidad de control.
2. Decodificar la instrucción; es decir, determinar la instrucción de máquina que debe ejecutarse.
3. Recuperar los operandos necesarios para la instrucción desde la memoria o desde los dispositivos de entrada.
4. Realizar la operación indicada por la instrucción.
5. Guardar en memoria los resultados generados por la operación, o enviar los resultados a un dispositivo de salida.

Las instrucciones se recuperan de la memoria en orden secuencial, a menos que se encuentre un tipo especial de instrucción, llamado indistintamente *ramificación*, *salto*, *brinco* o *transferencia*. Las instrucciones de ramificación permiten escribir programas con ciclos y toma de decisiones.

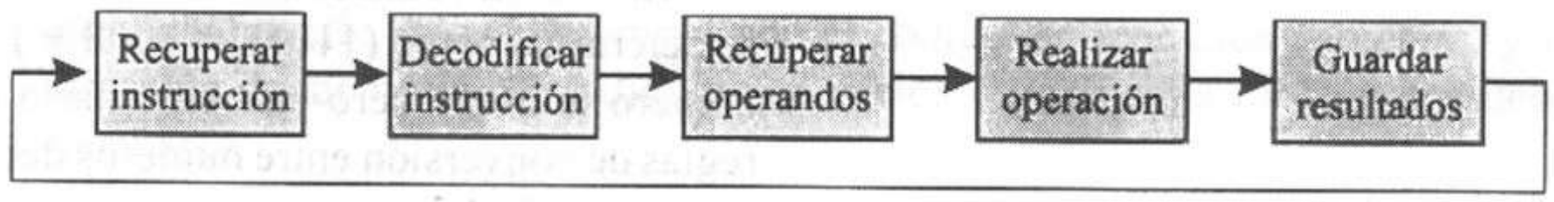


Figura 0.8 Ciclo de instrucción de una computadora de programa almacenado.

0.3.1 Instrucciones de la computadora

Cuando la unidad de control de una computadora digital recupera una instrucción desde la memoria para su ejecución, se pueden tener varios tipos de operaciones.

1. Las *instrucciones aritméticas* hacen que los datos binarios se sumen, resten, multipliquen o dividan según lo especificado por el programador de la computadora en el programa.

2. Se dispone de *operaciones de prueba y comparación* que determinan la relación (mayor que, menor que, igual a u otras) entre dos elementos de datos binarios.

3. Las *instrucciones de ramificación o de salto* se pueden utilizar para alterar la naturaleza secuencial de la ejecución del programa, con base en los resultados de una prueba o comparación. Este tipo de función añade una gran flexibilidad a los programas.

4. Se incluyen las *instrucciones de entrada y salida* para leer mensajes que llegan a la computadora, escribir mensajes desde la computadora y controlar dispositivos periféricos.

5. Las *operaciones lógicas y de corrimiento* permiten a la computadora traducir e interpretar todos los diferentes códigos que usa. Estas instrucciones permiten el manejo de bits bajo el control del programa.

Todas las instrucciones de cualquier computadora digital pueden agruparse en una de estas cinco categorías.

0.3.2 Representación de la información en las computadoras

Hemos analizado brevemente las instrucciones y datos almacenados en la unidad de memoria de la computadora digital, pero no hemos mencionado la forma de estos elementos. Por lo general, la información de un sistema de cómputo se puede dividir en tres categorías: datos numéricos, datos no numéricos y códigos de instrucciones.

Representación de datos numéricos

Los números se guardan en la memoria de la computadora con el sistema de numeración binario (base 2). Los números binarios se escriben con los dos dígitos binarios (bits) 1 y 0. En cambio, nosotros utilizamos 10 dígitos decimales al escribir los números decimales.

Por ejemplo, 129 en decimal significa $1 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$; es decir, la posición de cada dígito representa una potencia ponderada de 10. Observe que los 10 dígitos van de 0 a $10 - 1 = 9$. Cada dígito de un número binario, como 1101, se representa mediante una potencia ponderada de 2, como $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$. Para convertir el número binario a decimal, esta suma ponderada se determina como $(1101)_2 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = (13)_{10}$; es decir, el número uno-uno-cero-uno en binario es equivalente al 13 en decimal. Estas reglas de conversión entre números decimales y binarios se analizan con detalle en el capítulo 1.

Los datos en forma de números binarios se guardan en registros de la computadora y se representan como sigue:

1011000111

Éste es un registro de 10 bits, que podría residir en la unidad aritmética o en la memoria. En esta última, los datos de un registro son una *palabra* (en este ejemplo, la longitud de la palabra es de 10 bits). Los patrones de unos y ceros son la única información que puede guardarse en los registros o memoria de una computadora. La asignación de un significado a los patrones de bits es una *codificación*, y los códigos para datos que se utilizan en la mayor parte de las computadoras sólo son variantes del esquema binario ponderado que hemos presentado.

Códigos no numéricos (entrada/salida)

Aunque la computadora utiliza datos binarios, los usuarios prefieren la representación de datos alfabéticos y numéricos como, por ejemplo, registros de ventas, listas de nombres o calificaciones de exámenes. El conjunto de símbolos alfanuméricos permitidos por muchas computadoras es su conjunto de caracteres y tiene un código especial parecido al binario, llamado *Código Estándar Americano para Intercambio de Información* (ASCII). En este código, los caracteres alfanuméricos y otros caracteres especiales (puntuación, operadores algebraicos y otros) se codifican con 8 bits cada uno; en el capítulo 1 aparece una lista parcial de este código. Suponga que queremos enviar a la computadora digital el mensaje "ADD 1" (sumar 1). Este mensaje tiene cinco caracteres, donde el cuarto es un espacio o blanco. En el código ASCII, nuestro mensaje se convierte en

Símbolo	Código ASCII
A	01000001
D	01000100
D	01000100
	00100000
1	00110001

Después de enviar nuestro mensaje a la computadora, un programa en la memoria de ésta lo acepta y actúa de manera acorde.

Códigos de instrucciones

Las instrucciones de la computadora residen en la memoria principal y, por tanto, por definición, también se representan mediante patrones de unos y ceros. Por lo general, las instrucciones se descomponen en subcampos codificados individualmente. Estos subcampos son el código de operación (*op code*) y la dirección de memoria. El código de operación especifica la función que debe ejecutarse.

0.3.3 Hardware de computadora

Examinemos ahora con más detalle la interacción de los componentes de la computadora de la figura 0.7. Los programas se guardan en la memoria de la computadora, como hemos visto. Sin embargo, la unidad de control inserta los programas en la memoria auxiliada por el equipo de entrada/salida (E/S) o *dispositivos periféricos*. Por lo general, los programas se proporcionan a la computadora mediante dispositivos periféricos de almacenamiento magnético u óptico. Después, la computadora recupera las instrucciones del programa desde la memoria y las ejecuta. Los datos que utilizará un programa se transfieren del mismo modo a la memoria desde los teclados, digitalizadores, discos magnéticos y otros dispositivos periféricos.

Unidad de control

La unidad de control sigue la lista de instrucciones guardada y dirige las actividades de la unidad aritmética y de los dispositivos de E/S hasta que el programa termina su ejecución. Cada unidad realiza su tarea bajo la influencia sincronizadora de la unidad de control.

Unidad aritmética lógica

Las unidades aritmética lógicas (UAL) son circuitos lógicos combinatorios o secuenciales que realizan diversas operaciones sobre los datos, según lo indicado por la unidad de control. Cada UAL se caracteriza por el tipo de datos que puede manejar y por el conjunto de operaciones que puede realizar sobre tales datos. La mayor parte de las UAL pueden operar con enteros de varios tamaños y también puede incluir operaciones para manipular números de punto fijo, números de punto flotante y diversos datos no numéricos. Entre las operaciones de UAL típicas están:

- Aritméticas: suma, resta, multiplicación y división.
- Lógicas: AND, OR, OR exclusivo, complemento (definiremos estas operaciones cuando analicemos los circuitos lógicos combinatorios en el capítulo 2).
- Corrimiento y rotación de datos.
- Conversión de datos de un tipo a otro.

Por lo general, los circuitos de la unidad de control y de la UAL se construyen con dispositivos semiconductores en varios esquemas de empaque. Los modelos de la segunda generación tienen transistores, resistencias, diodos, etc., montados en tarjetas de circuitos impresos, mientras que los modelos de la

tercera generación utilizan circuitos integrados a pequeña escala en tarjetas de circuitos. Las máquinas de la cuarta generación utilizan circuitos integrados a gran escala y a muy grande escala.

Unidades de memoria

Las unidades de memoria de la computadora se clasifican como *memoria primaria* si la unidad de control tiene acceso directo a ellas; de otro modo, se clasifican como *memoria secundaria.*

Las unidades de memoria primaria en las computadoras digitales actuales se construyen con elementos semiconductores de alta velocidad llamados *RAM* (memoria de acceso aleatorio) y *ROM* (memoria exclusiva para lectura). Casi todos los sistemas construidos antes de 1980, algunos de los cuales todavía están en operación, utilizaron arreglos de núcleos magnéticos como elementos de memoria primaria. Unos cuantos sistemas especializados, en especial los de vehículos espaciales, utilizaron alambre chapeado como reemplazo del núcleo magnético en aplicaciones en las que se requería mayor resistencia a la radiación.

Las unidades de memoria están divididas en celdas llamadas *palabras*, y cada celda se identifica por su lugar físico, o dirección de memoria. El concepto de dirección de memoria para una celda de memoria es equivalente a una dirección de correo para un buzón. Por ejemplo, cada oficina de correo tiene filas de buzones, donde cada uno se identifica con una posición numerada única. De manera similar, cada celda de memoria reside en una posición numerada única, donde el número es la dirección de memoria.

Las unidades de memoria se pueden caracterizar según sus tiempos de acceso y de ciclo; el *tiempo de acceso a la memoria* se puede definir como el intervalo de tiempo necesario para extraer (leer) una palabra de la memoria, y el *tiempo de ciclo de memoria* se puede definir como el intervalo mínimo de tiempo necesario entre dos operaciones sucesivas de la memoria. El tiempo de acceso de una memoria determina la rapidez con que la CPU puede obtener la información, mientras que el tiempo de ciclo determina la velocidad con que se pueden hacer accesos sucesivos a la memoria.

Los dispositivos de memoria secundaria se utilizan para el almacenamiento en masa de programas y datos, e incluyen ciertos dispositivos magnéticos giratorios, como los discos flexibles o duros, las cintas magnéticas, las memorias de burbuja magnética, los dispositivos ópticos como los CD-ROM (discos compactos exclusivos para lectura) y una gran variedad de dispositivos. En contraste con la memoria primaria, no hay un acceso directo a la información en los dispositivos de memoria secundaria, sino que un controlador especial busca en el dispositivo el bloque de información que contiene el elemento deseado. Cuando lo encuentra, por lo general se transfiere todo el bloque a la memoria primaria, donde el acceso a los elementos deseados se realiza de una manera más conveniente.

Equipo de entrada/salida

La computadora puede enviar datos hacia diversos tipos de periféricos, comúnmente los discos magnéticos o las impresoras láser. También se dispone de pantallas de tubo de rayos catódicos (CRT) y de cristal líquido (LCD) para presentar los resultados de los cálculos de un programa. Los convertidores analógico a digital y digital a analógico, los graficadores, los dispositivos de lectura y almacenamiento magnético y las impresoras láser y de inyección de tinta son los equipos de entrada/salida de uso más común.

0.3.4 Software de computadora

El software consiste en los programas y datos almacenados en la memoria de la computadora. El software determina la forma de utilizar el hardware de la computadora y se puede clasificar de manera amplia como programas de aplicación o programas de sistema.

Programas de aplicación

La programación de la computadora digital es el proceso de diseño de una lista de instrucciones para la computadora, de modo que ésta pueda realizar de manera eficiente una tarea específica. Las instrucciones de una computadora digital deben codificarse en patrones de unos y ceros antes de que ésta las pueda interpretar. Si todos los programas debieran escribirse en esta forma, las computadoras digitales tendrían un uso muy limitado. Los patrones de unos y ceros son las instrucciones de lenguaje de máquina, y muy pocos programadores han intentado escribir programas de esta manera.

Con frecuencia se utiliza una representación simbólica del lenguaje de máquina de una computadora, llamada *lenguaje de ensamblador*, para el desarrollo de los programas. Esto ocurre en particular con las microcomputadoras pequeñas de los aparatos de cocina, juegos electrónicos y equipo automotriz. El lenguaje de ensamblador permite a un programador especificar las operaciones por realizar sobre los datos almacenados en los registros y memoria internos de un procesador sin empantanarse en los patrones de unos y ceros.

Sin embargo, la mayoría de los programadores prefiere utilizar lenguajes simbólicos de alto nivel, más razonables, para programar sus problemas. Con lenguajes de alto nivel como C, Pascal, Ada o FORTRAN, el programador dispone de una amplia gama de instrucciones que puede comprender con facilidad y utilizar con eficiencia, de modo que las instrucciones de cada lenguaje se ajustan a tipos específicos de problemas. Es difícil incorporar toda la flexibilidad del lenguaje de máquina en los lenguajes de alto nivel, pero se ha mantenido en una proporción significativa, sobre todo en C.

Programas de sistema

Los *programas de sistema* incluyen todo el software incluido en un sistema de cómputo para ayudar a los programadores en el proceso de desarrollo y ejecución de programas de aplicación. Por ejemplo, cuando un lenguaje simbólico, ya sea de ensamblador o de alto nivel, se utiliza para escribir un programa, éste debe traducirse al lenguaje de máquina antes de que la computadora pueda ejecutarlo. Surge la pregunta de quién debe realizar esta laboriosa traducción. El traductor más eficiente no es el programador, sino la propia computadora digital. Cualquier trabajo realizado por una computadora está bajo el control de un programa; por tanto, el programa que traduce los lenguajes de alto nivel al lenguaje de máquina tiene un nombre especial, el *compilador*. Este proceso de traducción se ilustra en la figura 0.9. De manera análoga, el programa que traduce el lenguaje de ensamblador al lenguaje de máquina es un *ensamblador*. Los compiladores y los ensambladores son ejemplos típicos de programas de sistema, al igual que los editores de textos que se utilizan para escribir y alterar enunciados de programa.

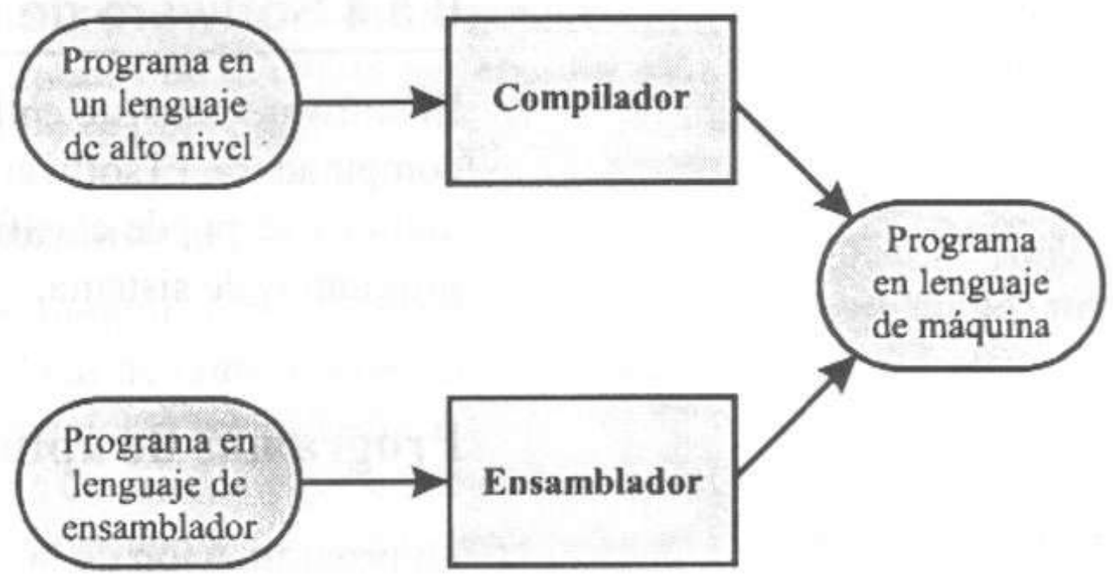


Figura 0.9 Traducción de programas de computadora a lenguaje de máquina.

Ahora, veamos la manera en que podría operar una computadora digital. Un programa especial llamado *sistema operativo* controla las tareas rutinarias necesarias para pasar de un programa de usuario al siguiente. Este programa especial pertenece a una categoría de programas llamada software de sistemas. En particular, analizaremos tres sistemas operativos diferentes: para un único usuario, para procesamiento por lotes y de tiempo compartido.

Si una máquina es operada por cada usuario que ejecuta un programa, la computadora se dedica a este programa y nadie más puede utilizarla en tanto el usuario actual no haya terminado. La computadora depende entonces de la intervención humana durante el tiempo entre programas, y podría pasar mucho tiempo inactiva. Este sistema operativo es conveniente para el usuario si la computadora está disponible cuando el usuario la necesita, ya que, si el individuo está "en la máquina", puede modificar y volver a ejecutar programas o ejecutar varios programas sucesivos antes de dejar la máquina para otro usuario. Los sistemas MS-DOS y Macintosh son ejemplos de sistemas operativos de usuario único.

En los sistemas operativos de procesamiento por lotes se elimina la mayor parte del tiempo de inactividad de la computadora, al designar a un operador que reúne todos los programas de usuario y los alimenta a la computadora en forma continua. El programa del sistema operativo reside en la unidad de memoria, y las localidades de memoria que utiliza están protegidas de los usuarios. Así, aunque se reduce el tiempo de inactividad, el espacio de almacenamiento de memoria disponible para el usuario también se reduce. Además, el usuario debe esperar a que el operador devuelva su programa, lo que siempre es una fuente de irritación y enfrentamiento.

Un sistema operativo más avanzado, llamado de *tiempo compartido,* permite que varios usuarios ejecuten sus programas casi en forma simultánea. Los ejemplos comunes son el sistema operativo UNIX, utilizado en una amplia variedad de computadoras personales, estaciones de trabajo y máquinas de mayor tamaño, y el sistema operativo VMS, empleado en computadoras de la Digital Equipment Corporation. Las terminales remotas, integradas por dispositivos de entrada/salida limitados, están conectadas a la computadora digital, y cada terminal es asignada a un único usuario. Los usuarios son relativamente lentos, mientras que la computadora es muy rápida. Esta diferencia de velocidades permite a la computadora ir de un usuario a otro, compartiendo el tiempo, de modo que puede convencer a cada usuario de que tiene toda la máquina sólo para él. Aunque este sistema operativo parece muy atractivo, tiene sus

desventajas, la primera de las cuales es el costo. Además, el programa de sistema de tiempo compartido es complejo y de gran tamaño, lo que significa que utiliza mucho espacio de memoria y tiempo de cómputo. Como todos los programas de usuario se guardan en la memoria de manera simultánea, la porción de memoria disponible para cada individuo es limitada. Por tanto, el tiempo compartido requiere el número máximo de elementos de memoria que una computadora particular pueda proporcionar.

0.4 Resumen

En este capítulo introductorio hemos ofrecido una motivación para estudiar el material posterior. Hemos explicado en forma breve lo que es una computadora, cómo está organizada, qué códigos utiliza, cómo puede programarse, y qué hardware la compone. El material del resto de los capítulos es necesario para cualquier diseño o implantación de hardware para computadoras digitales u otros sistemas digitales complejos. El lector puede recurrir a [1, 2, 3, 4] como lecturas adicionales.

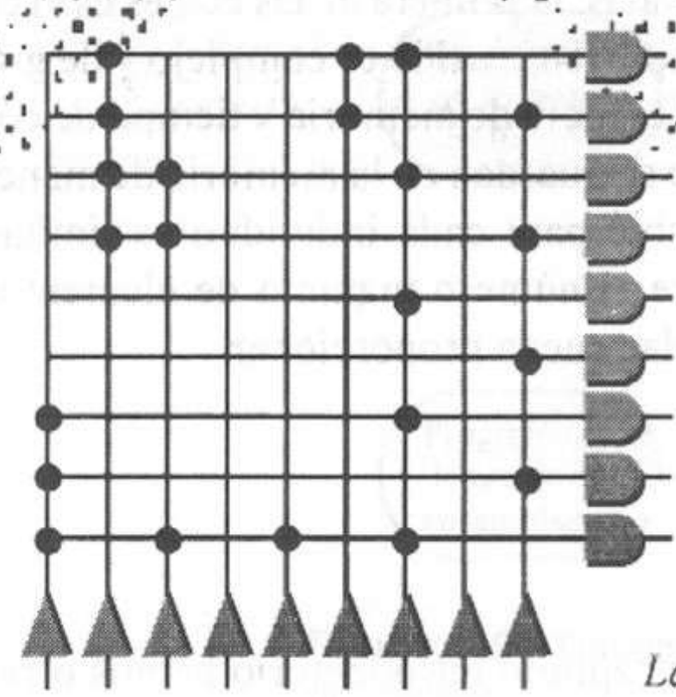
REFERENCIAS BIBLIOGRÁFICAS

1. J. P. Hayes, *Computer Architecture and Organization*, segunda edición. Nueva York: McGraw-Hill Book Co., 1988.

2. D. A. Patterson y J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*. San Mateo, CA: Morgan Kaufmann Publishers, 1993

3. D. A. Hodges y H. G. Jackson, *Analysis and Design of Digital Integrated Circuits*, segunda edición. Nueva York: McGraw-Hill Book Co., 1988.

4. J. F. Wakerly, *Digital Design Principles and Practices*, segunda edición. Englewood Cliffs, NJ: Prentice Hall, 1994.



La función principal de las computadoras y otros sistemas digitales es la de procesar información. Por tanto, es necesario contar con métodos y sistemas para representar la información en formas que se puedan manipular y almacenar utilizando equipo electrónico o de otro tipo.

En este capítulo analizaremos los sistemas numéricos y los códigos que se utilizan con frecuencia en las computadoras y los sistemas digitales. Los temas incluyen los sistemas numéricos binario, octal y hexadecimal y su aritmética; las técnicas de conversión de base; los métodos para representar números negativos, como los de magnitud y signo, complemento a dos y complemento a uno; los códigos numéricos para números de punto fijo y flotante; los códigos de caracteres, incluidos el decimal codificado en binario y el ASCII; los códigos Gray y de exceso; y los códigos para detección y corrección de errores. Los capítulos posteriores del libro tratan el análisis y diseño de hardware para procesar información representada en las formas aquí descritas.

Sistemas numéricos y códigos

1.1 Sistemas numéricos

Un sistema numérico consta de un conjunto ordenado de símbolos, llamados *dígitos*, con relaciones definidas para la suma (+), resta (−), multiplicación (×) y división (÷). La *base* (r) del sistema numérico es el número total de dígitos permitidos en dicho sistema. Los sistemas numéricos de uso común en el diseño de sistemas digitales y la programación de computadoras incluyen el *decimal* ($r = 10$), el *binario* ($r = 2$), el *octal* ($r = 8$) y el *hexadecimal* ($r = 16$). Cualquier número en un sistema dado puede tener una parte entera y una parte fraccionaria, que se separan mediante un punto (.). En algunos casos, puede faltar la parte entera o la parte fraccionaria. Ahora, examinaremos las notaciones *posicional* y *polinomial* de un número.

1.1.1 Notación posicional

Suponga que pide a su banco local un préstamo por ciento veintitrés dólares y treinta y cinco centavos. El cheque que le dan indica la cantidad como $123.35. Al escribir este número, se ha utilizado la notación posicional. El cheque puede cobrarse con un billete de cien dólares, dos billetes de diez dólares, tres billetes de un dólar, tres monedas de diez centavos y cinco monedas de un centavo. Por tanto, la posición de cada dígito indica su peso o significado relativo.

En general, un número positivo N se puede escribir en notación posicional como

$$N = (a_{n-1}a_{n-2}...a_1a_0 \,.\, a_{-1}a_{-2}...a_{-m})_r \tag{1.1}$$

donde

. = punto que separa los dígitos enteros y fraccionarios
r = base del sistema numérico que se está utilizando
n = número de dígitos enteros a la izquierda del punto
m = número de dígitos fraccionarios a la derecha del punto
a_i = dígito entero i cuando $n - 1 \geq i \geq 0$
a_i = dígito fraccionario i cuando $-1 \geq i \geq -m$
a_{n-1} = dígito más significativo
a_{-m} = dígito menos significativo

Observe que el intervalo de valores para los dígitos a_i es $r-1 \geq a_i \geq 0$. Con esta notación, la cantidad del préstamo bancario podría escribirse $\$(123.35)_{10}$. Los paréntesis y el subíndice que denota la base pueden eliminarse sin perder información siempre que la base se conozca por el contexto o se especifique de otra forma.

Notación polinomial

Podemos escribir la cantidad del préstamo de $(123.35)_{10}$ dólares en forma polinomial como

$$N = 1 \times 100 + 2 \times 10 + 3 \times 1 + 3 \times 0.1 + 5 \times 0.01$$
$$= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 3 \times 10^{-1} + 5 \times 10^{-2}$$

Observe que cada dígito está en una posición ponderada y que el peso de cada posición es una potencia de la base, 10. En general, cualquier número N con base r se puede escribir como un polinomio de la forma

$$N = \sum_{i=-m}^{n-1} a_i r^i \tag{1.2}$$

donde cada símbolo se define como en la ecuación 1.1. Para el préstamo del banco, $r = 10$, $a_2 = 1$, $a_1 = 2$, $a_0 = 3$, $a_{-1} = 3$, $a_{-2} = 5$ y $a_i = 0$, para $i \geq 3$ y para $i \leq -3$.

1.1.2 Sistemas numéricos de uso común

Los sistemas numéricos decimal, binario, octal y hexadecimal son importantes para el estudio de los sistemas digitales. La tabla 1.1 resume las características fundamentales de cada sistema e ilustra un intervalo limitado de enteros positivos en cada uno. Todos los números de la tabla 1.1 están escritos en notación posicional.

Por lo general, los sistemas digitales se construyen con dispositivos de dos estados (apagado o encendido). Por tanto, el sistema numérico binario es ideal para representar números en sistemas digitales, ya que sólo se necesitan dos dígitos, 0 y 1, llamados *bits*. Un bit puede guardarse en un dispositivo de almacenamiento de dos estados conocido como *latch*. Los números binarios de longitud n se pueden guardar en un dispositivo de n bits de longitud, llamado *registro*, construido con n latches. Un registro de 8 bits que contiene el número binario 10011010 se muestra en la figura 1.1.

1	0	0	1	1	0	1	0

Figura 1.1 Un registro de 8 bits.

1.2 Aritmética

Todos los niños aprenden los rudimentos de la aritmética memorizando las tablas de suma y multiplicación en base 10 que aparecen en las tablas 1.2a y b, respectivamente. La resta se puede realizar utilizando la tabla de la suma al revés. De manera análoga, la división utiliza la multiplicación y la resta mediante prueba y error para obtener el cociente. El fundamento de la aritmética en cualquier base es el conocimiento de tablas de suma y multiplicación en la base dada. Dadas estas tablas, las operaciones aritméticas son similares para todas las bases. En el resto de esta sección presentaremos la aritmética en los sistemas numéricos binario, octal y hexadecimal.

TABLA 1.1 SISTEMAS NUMÉRICOS IMPORTANTES

Nombre	Decimal	Binario	Octal	Hexadecimal
Base	10	2	8	16
Dígitos	0,1,2,3,4, 5,6,7,8,9	0,1	0,1,2,3, 4,5,6,7	0,1,2,3,4,5, 6,7,8,9,*A,B, C,D,E,F*
Primeros	0	0	0	0
diecisiete	1	1	1	1
positivos	2	10	2	2
enteros	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	10000	20	10

1.2.1 Aritmética binaria

Suma

Las tablas 1.3a y b muestran las tablas de suma y multiplicación, respectivamente, para el sistema numérico binario. Las tablas son muy pequeñas ya que sólo hay dos dígitos, o *bits*, en el sistema. En consecuencia, la aritmética binaria es muy sencilla. Observe que la suma 1+1 produce un bit de suma de 0 y un bit de acarreo de 1. El acarreo debe sumarse a la siguiente columna de bits para realizar la suma en el patrón normal, de derecha a izquierda. Daremos dos ejemplos de suma binaria a continuación.

EJEMPLO 1.1

Sumar los dos números binarios $(111101)_2$ y $(10111)_2$.

	1	*1*	*1*	*1*	*1*	*1*		*Acarreos*
		1	1	1	0	0	1	Sumando
+			1	0	1	1	1	Sumando
	1	0	1	0	1	0	0	Suma

TABLA 1.2 (a) TABLA DE SUMA DECIMAL; (b) TABLA DE MULTIPLICACIÓN DECIMAL.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

(a)

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	9	9	18	27	36	45	54	63	72	81

(b)

TABLA 1.3 (a) TABLA DE SUMA BINARIA; (b) TABLA DE MULTIPLICACIÓN BINARIA.

+	0	1
0	0	1
1	1	10

(a)

×	0	1
0	0	0
1	0	1

(b)

En el ejemplo 1.1 hay dos columnas que tienen dos bits 1 y un bit de acarreo 1, lo que debe sumarse. Esta suma de tres unos puede verse más fácilmente como

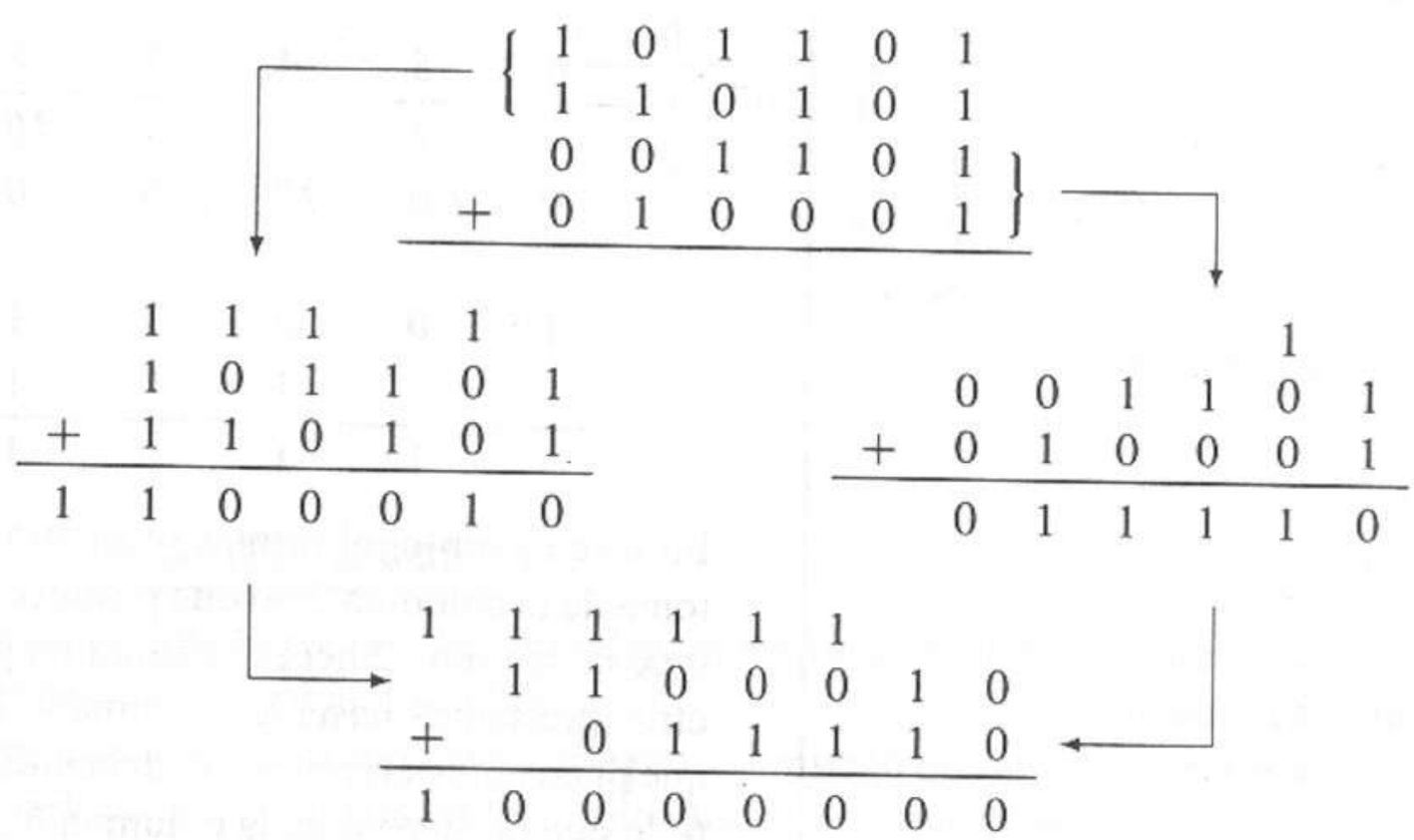
$$\begin{aligned} 1 + 1 + 1 &= (1 + 1) + 1 \\ &= (10)_2 + (01)_2 \\ &= 11 \end{aligned}$$

Así, los bits de suma y de acarreo son ambos 1.

Cuando hay que sumar una gran lista de números binarios, los cálculos se realizan fácilmente si sumamos los números por pares, como se demuestra en el siguiente ejemplo.

EJEMPLO 1.2

Sumar los cuatro números $(101101)_2$, $(110101)_2$, $(001101)_2$ y $(010001)_2$.



Sin embargo, podemos optar por sumar en forma directa para evitar los pasos intermedios necesarios en el método anterior. El método directo se ilustra en el siguiente ejemplo.

EJEMPLO 1.3

Repetir el ejemplo anterior sumando toda una columna a la vez.

	10	*10*	*10*	*10*	*1*	*10*		*Acarreos*
		1	0	1	1	0	1	
		1	1	0	1	0	1	
		0	0	1	1	0	1	
+		0	1	0	0	0	1	
1	0	0	0	0	0	0	0	Suma

Observe que la suma de los dígitos en la primera columna es $1 + 1 + 1 + 1 = (100)_2$. Esto produce un dígito de suma igual a 0 en esa columna y un acarreo de 10 en la siguiente columna a la izquierda.

Resta

La resta se puede visualizar como el inverso de la suma. Las reglas para la resta binaria se derivan directamente de la tabla de suma binaria (tabla 1.3a) y son

$$1 - 0 = 1$$
$$1 - 1 = 0$$

$0 - 0 = 0$

$0 - 1 = 1$ tomando prestado 1, o $10 - 1 = 1$

La última regla muestra que si se resta un bit 1 de un bit 0, hay que tomar prestado un 1 de la siguiente columna más significativa. Los préstamos se propagan hacia la izquierda de columna en columna, como se ilustra a continuación.

EJEMPLO 1.4

Restar $(10111)_2$ a $(1001101)_2$.

6	5	4	3	2	1	0	Columna
	1			*10*			*Préstamos*
0	*~~1~~ ~~0~~*	*10*	*0*	*~~0~~*	*10*		*Préstamos*
~~1~~	*~~0~~*	*~~0~~*	*~~1~~*	*~~1~~*	*~~0~~*	1	Minuendo
–		1	0	1	1	1	Sustraendo
	1	1	0	1	1	0	Diferencia

En este ejemplo, el primer préstamo aparece en la columna 1. El préstamo se toma de la columna 2, lo que produce 10 en la columna 1 y 0 en la columna 2. El 0 de la columna 2 necesita ahora un préstamo de la columna 3. No se necesitan otros préstamos hasta la columna 4. En este caso, no hay un 1 en la columna 5 que pueda prestar; por tanto, debemos primero tomar prestado el 1 de la columna 6, lo que produce 0 en la columna 6 y 10 en la columna 5. Ahora, la columna 4 toma prestado 1 de la columna 5, lo que deja 1 en la columna 5 $(10 - 1 = 1)$ y 10 en la columna 4. Esta serie de préstamos se muestra arriba de los términos del minuendo.

Multiplicación y división

La multiplicación binaria se realiza en forma similar a la multiplicación decimal, excepto que las operaciones de multiplicación binaria son mucho más sencillas, como se puede ver en la tabla 1.3b. No obstante, se debe tener mucho cuidado al sumar los productos parciales, como se ilustra en el siguiente ejemplo.

EJEMPLO 1.5

Multiplicar $(10111)_2$ por $(1010)_2$.

			1	0	1	1	1	Multiplicando
×				1	0	1	0	Multiplicador
			0	0	0	0	0	
		1	0	1	1	1		
	0	0	0	0	0			
1	0	1	1	1				
1	1	1	0	0	1	1	0	Producto

Observe que hay un producto parcial por cada bit del multiplicador. Este procedimiento puede realizarse con mayor eficiencia si sólo recorremos una columna a la izquierda, en vez de anotar un producto parcial con ceros para un bit 0 del multiplicador. Este ejemplo nos sirve para ver lo sencillo de este procedimiento.

La división binaria se realiza utilizando el mismo procedimiento de prueba y error de la división decimal. Sin embargo, la división binaria es más sencilla pues sólo hay que intentar con dos valores. Se restan del dividendo copias de los términos del divisor, de lo cual se obtienen residuos intermedios positivos. El siguiente ejemplo ilustra la división binaria.

EJEMPLO 1.6

Dividir $(1110111)_2$ entre $(1001)_2$.

								1	1	0	1	Cociente
Divisor	1	0	0	1	1	1	1	0	1	1	1	Dividendo
					1	0	0	1				
						1	0	1	1			
						1	0	0	1			
								1	0	1	1	
								1	0	0	1	
										1	0	Residuo

1.2.2 Aritmética octal

Las tablas de la suma y la multiplicación para el sistema de numeración octal aparecen en la tabla 1.4. Con la ayuda de estas tablas, la aritmética octal se puede realizar utilizando el mismo procedimiento que en los sistemas decimal y binario, como se ilustra en los siguientes cuatro ejemplos.

EJEMPLO 1.7

Calcular $(4163)_8 + (7520)_8$.

1		1			Acarreos
	4	1	6	3	Sumando
+	7	5	2	0	Sumando
1	3	7	0	3	Suma

EJEMPLO 1.8

Calcular $(6204)_8 - (5173)_8$.

		1	10		Prestado
	6	~~2~~	~~0~~	4	Minuendo
−	5	1	7	3	Sustraendo
	1	0	1	1	Diferencia

EJEMPLO 1.9

Calcular $(4167)_8 \times (2503)_8$.

					4	1	6	7	Multiplicando
				×	2	5	0	3	Multiplicador
				1	4	5	4	5	Productos parciales
	2	5	1	2	3	0			
1	0	3	5	6					
1	3	1	0	5	0	4	5		Producto

TABLA 1.4 (a) TABLA DE SUMA OCTAL; (b) TABLA DE MULTIPLICACIÓN OCTAL

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

(a)

X	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

(b)

EJEMPLO 1.10

Calcular $(4163)_8 \div (25)_8$.

```
                    1   4   7   Cociente
Divisor   2   5 | 4   1   6   3   Dividendo
                  2   5
                  ---
                  1   4   6
                  1   2   4
                  -------------
                      2   2   3
                      2   2   3
                      -------------
                              0   Residuo
```

1.2.3 Aritmética hexadecimal

Las tablas de la suma y la multiplicación hexadecimales son más complejas que las de los sistemas numéricos estudiados anteriormente, y aparecen en la tabla 1.5. De cualquier manera, al igual que en los otros sistemas numéricos, el

TABLA 1.5 TABLAS DE SUMA Y MULTIPLICACIÓN HEXADECIMAL (a) TABLA DE SUMA HEXADECIMAL (b) TABLA DE MULTIPLICACIÓN HEXADECIMAL

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

(a)

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

(b)

conocimiento de estas tablas permite realizar la aritmética hexadecimal con procedimientos bien conocidos. Los siguientes cuatro ejemplos ilustran la aritmética hexadecimal.

EJEMPLO 1.11

Calcular $(2A58)_{16} + (71D0)_{16}$.

		1			Acarreos
	2	*A*	5	8	Sumando
+	7	1	*D*	0	Sumando
	9	*C*	2	8	Suma

EJEMPLO 1.12

Calcular $(9F1B)_{16} - (4A36)_{16}$.

		E	11		Préstamos
	9	~~*F*~~	~~1~~	*B*	Minuendo
−	4	*A*	3	6	Sustraendo
	5	4	*E*	5	Diferencia

EJEMPLO 1.13

Calcular $(5C2A)_{16} \times (71D0)_{16}$.

					5	*C*	2	*A*	Multiplicando
				×	7	1	*D*	0	Multiplicador
			4	*A*	*E*	2	2	0	Productos parciales
			5	*C*	2	*A*			
2	8	5	2	6					
2	8	*F*	9	6	*C*	2	0		Producto

EJEMPLO 1.14

Calcular $(27FCA)_{16} \div (3E)_{16}$.

				A	5	1	Cociente
Divisor 3	*E*	2	7	*F*	*C*	*A*	Dividendo
		2	6	*C*			
			1	3	*C*		
			1	3	6		
					6	*A*	
					3	*E*	
					2	*C*	Residuo

1.3 Conversiones de base

Con frecuencia, los usuarios y diseñadores de computadoras y otros sistemas digitales tienen necesidad de convertir un número dado en base *A* a su equivalente en base *B*. En esta sección presentamos e ilustramos los algoritmos para realizar conversiones de base.

1.3.1 Métodos de conversión

Sustitución de una serie

La representación polinomial de un número dado por la ecuación 1.2 es la base del método de conversión por *sustitución de una serie*. La ecuación puede expandirse como sigue:

$$N = a_{n-1}r^{n-1} + \ldots + a_0r^0 + a_{-1}r^{-1} + \ldots + a_{-m}r^{-m} \tag{1.3}$$

Un número en base A puede convertirse en un número en base B en dos pasos.

1. Se forma la representación en serie del número en base A en el formato de la ecuación 1.3.
2. Se evalúa la serie utilizando la aritmética de base B.

Los siguientes cuatro ejemplos ilustran este procedimiento.

EJEMPLO 1.15

Convertir $(10100)_2$ a base 10.

Realizamos esta conversión sustituyendo cada dígito de acuerdo con su peso. Si contamos de derecha a izquierda en $(10100)_2$, vemos que el dígito del extremo derecho, 0, tiene un peso de 2^0, el siguiente dígito, 2^1, y así sucesivamente. Al sustituir estos valores en la ecuación 1.3 y evaluar la serie con aritmética de base 10 obtenemos

$$\begin{aligned} N &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ &= (16)_{10} + 0 + (4)_{10} + 0 + 0 \\ &= (20)_{10} \end{aligned}$$

EJEMPLO 1.16

Convertir $(274)_8$ a base 10.

$$\begin{aligned} N &= 2 \times 8^2 + 7 \times 8^1 + 4 \times 8^0 \\ &= (128)_{10} + (56)_{10} + (4)_{10} \\ &= (188)_{10} \end{aligned}$$

EJEMPLO 1.17

Convertir $(1101.011)_2$ a base 8.

La parte entera del número se convierte como en los ejemplos anteriores. Con los dígitos que están a la derecha del punto binario, contamos de izquierda a derecha. El primer dígito a la derecha del punto binario, 0, tiene un peso de 2^{-1}, el siguiente dígito, 1, tiene un peso de 2^{-2} y el tercer dígito, 1, tiene un peso de 2^{-3}. Al sustituir en la ecuación 1.3 obtenemos

$$\begin{aligned} N &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= (10)_8 + (4)_8 + 0 + (1)_8 + 0 + (.2)_8 + (.1)_8 \\ &= (15.3)_8 \end{aligned}$$

EJEMPLO 1.18

Convertir $(AF3.15)_{16}$ a base 10.

$$
\begin{aligned}
N &= A \times 16^2 + F \times 16^1 + 3 \times 16^0 + 1 \times 16^{-1} + 5 \times 16^{-2} \\
&= 10_{10} \times 256_{10} + 15_{10} \times 16_{10} + 3_{10} \times 1_{10} \\
&\quad + 1_{10} \times 0.0625_{10} + 5_{10} \times 0.00390625_{10} \\
&= 2560_{10} + 240_{10} + 3_{10} + 0.0625_{10} + 0.01953125_{10} \\
&= (2803.08203125)_{10}
\end{aligned}
$$

En los ejemplos anteriores, observe que los cálculos fueron más sencillos para las conversiones de base A a base B cuando $A < B$. Ahora describiremos algunos métodos de conversión en donde es cierto lo contrario.

Método de división entre la base

El método de conversión de *división entre la base* se puede utilizar para convertir un entero en base A al entero equivalente en base B. Para comprender el método, consideremos la siguiente representación de un entero N_I.

$$(N_I)_A = b_{n-1}B^{n-1} + \ldots + b_0B^0 \tag{1.4}$$

En la ecuación 1.4, los b_i representan los dígitos de $(N_I)_B$ en base A. Podemos determinar el dígito menos significativo, $(b_0)_A$, al dividir $(N_I)_A$ entre $(B)_A$ como sigue:

$$
\begin{aligned}
N_I/B &= (b_{n-1}B^{n-1} + \ldots + b_1B^1 + b_0B^0)/B \\
&= \underbrace{b_{n-1}B^{n-2} + \ldots + b_1B^0}_{\text{Cociente, } Q_1} + \underbrace{b_0}_{\text{Residuo, } R_0}
\end{aligned}
$$

En otras palabras, $(b_0)_A$ es el residuo obtenido al dividir $(N_I)_A$ entre $(B)_A$. En general, $(b_i)_A$ es el residuo R_i obtenido al dividir el cociente Q_i entre $(B)_A$. La conversión termina al convertir cada $(b_i)_A$ a base B. Sin embargo, este último paso es trivial si $B < A$. El procedimiento para la conversión mediante la división entre la base se resume como sigue.

1. Dividimos $(N_I)A$ entre la base deseada $(B)A$, obteniendo el cociente Q_1 y el residuo R_0. R_0 es el dígito menos significativo, d_0, del resultado.
2. Calculamos cada uno de los dígitos restantes, d_i, para $i = 1 \ldots n\text{-}1$, dividiendo el cociente Q_i entre $(B)_A$ para obtener el cociente Q_{i+1} y el residuo R_i, que representa a d_i.
3. Nos detenemos cuando el cociente $Q_{i+1} = 0$.

El método de división entre la base se ilustra en los siguientes dos ejemplos.

EJEMPLO 1.19

Convertir $(234)_{10}$ a base 8.

Resolvemos este problema dividiendo repetidamente el entero $(234)_{10}$ (es decir, $(N)_A$, entre 8 (es decir, $(B)_A$, hasta que el cociente sea 0.

$$
\begin{array}{r|rrr}
 & & 2 & 9 \\
\hline
8 & 2 & 3 & 4 \\
 & 1 & 6 & \\
\hline
 & & 7 & 4 \\
 & & 7 & 2 \\
\hline
 & & & 2 \quad = b_0
\end{array}
\qquad
\begin{array}{r|rr}
 & & 3 \\
\hline
8 & 2 & 9 \\
 & 2 & 4 \\
\hline
 & & 5 \quad = b_1
\end{array}
\qquad
\begin{array}{r|r}
 & 0 \\
\hline
8 & 3 \\
 & 0 \\
\hline
 & 3 \quad = b_2
\end{array}
$$

Por tanto, $(234)_{10} = (352)_8$. Estos cálculos se pueden resumir en el siguiente formato breve:

$$
\begin{array}{r|rrr|r}
8 & 2 & 3 & 4 & 2 \\
\hline
 & 8 & 2 & 9 & 5 \\
\hline
 & & 8 & 3 & 3 \\
\hline
 & & & 0 &
\end{array}
\qquad
\begin{array}{l}
\text{LSB} \\
\\
\text{MSB}
\end{array}
$$

EJEMPLO 1.20

Convertir $(234)_{10}$ a base 16.

$$
\begin{array}{r|rrr}
 & & 1 & 4 \\
\hline
16 & 2 & 3 & 4 \\
 & 1 & 6 & \\
\hline
 & & 7 & 4 \\
 & & 6 & 4 \\
\hline
 & & 1 & 0 \quad = (A)_{16} = b_0
\end{array}
\qquad
\begin{array}{r|rr}
 & & 0 \\
\hline
16 & 1 & 4 \\
 & & 0 \\
\hline
 & 1 & 4 \quad = (E)_{16} = b_1
\end{array}
$$

Por tanto, $(234)_{10} = (EA)_{16}$. En notación breve:

$$
\begin{array}{r|rrr|l}
16 & 2 & 3 & 4 & 10 = (A)_{16} \\
\hline
 & 16 & 1 & 4 & 14 = (E)_{16} \\
\hline
 & & & 0 &
\end{array}
$$

Método de multiplicación por la base

Las conversiones de base para fracciones pueden realizarse mediante el método de *multiplicación por la base*. Sea NF una fracción en base A. La fracción se puede escribir como una serie:

$$(N_F)_A = b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m} \tag{1.5}$$

Los b_i de la ecuación 1.5 representan los dígitos de $(N_F)_B$ en base A. El dígito más significativo $(b_{-1})_A$ se puede obtener al multiplicar $(N_F)_A$ por $(B)_A$ como sigue:

$$\begin{aligned} B \times N_F &= B \times (b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m}) \\ &= \underbrace{b_{-1}}_{Entero,\ I_{-1}} + \underbrace{b_{-2}B^{-1} + \ldots + b_{-m}B^{-(m-1)}}_{Fracción,\ F_{-2}} \end{aligned}$$

Así, $(b_{-1})_A$ es la parte entera del producto que resulta de multiplicar $(N_F)_A$ por $(B)_A$. En general, $(b_{-i})_A$ es la parte entera I_{-i} del producto que resulta de multiplicar la fracción $F_{-(i+1)}$ por $(B)_A$. Por tanto, el procedimiento de multiplicación por la base se resume como sigue:

1. Sea $F_{-1} = (N_F)_A$.
2. Calculamos los dígitos $(b_{-i})_A$, para $i = 1 \ldots m$, multiplicando F_i por $(B)_A$ para obtener el entero I_{-i} que representa al dígito $(b_{-i})_A$ y la fracción $F_{-(i+1)}$.
3. Convertimos cada dígito $(b_{-i})_A$ a la base B.

Los siguientes dos ejemplos ilustran este método.

EJEMPLO 1.21

Convertir $(0.1285)_{10}$ a base 8.

$$\begin{array}{r} 0.1285 \\ \times \quad 8 \\ \hline 1.0280 \\ \uparrow \\ b_{-1} \end{array} \qquad \begin{array}{r} 0.0280 \\ \times \quad 8 \\ \hline 0.2240 \\ \uparrow \\ b_{-2} \end{array} \qquad \begin{array}{r} 0.2240 \\ \times \quad 8 \\ \hline 1.7920 \\ \uparrow \\ b_{-3} \end{array} \qquad \begin{array}{r} 0.7920 \\ \times \quad 8 \\ \hline 6.3360 \\ \uparrow \\ b_{-4} \end{array}$$

$$\begin{array}{r} 0.3360 \\ \times \quad 8 \\ \hline 2.6880 \\ \uparrow \\ b_{-5} \end{array} \qquad \begin{array}{r} 0.6880 \\ \times \quad 8 \\ \hline 5.5040 \\ \uparrow \\ b_{-6} \end{array} \qquad \begin{array}{r} 0.5040 \\ \times \quad 8 \\ \hline 4.0320 \\ \uparrow \\ b_{-7} \end{array} \qquad \begin{array}{r} 0.0320 \\ \times \quad 8 \\ \hline 0.2560 \\ \uparrow \\ b_{-8} \end{array}$$

Así,

$$0.1285_{10} = (0.10162540\ldots)_8$$

EJEMPLO 1.22

Convertir $(0.828125)_{10}$ a base 2.

Utilizaremos una notación abreviada en este ejemplo al aplicar el método de multiplicación por la base. En cada línea, la fracción se multiplica por 2 para obtener la línea siguiente:

$$
\begin{array}{r|l}
\text{MSD} & 1.656250 \leftarrow 0.828125 \times 2 \\
 & 1.312500 \leftarrow 0.656250 \times 2 \\
 & 0.625000 \leftarrow 0.312500 \times 2 \\
 & 1.250000 \leftarrow 0.625000 \times 2 \\
 & 0.500000 \leftarrow 0.250000 \times 2 \\
\text{LSD} & 1.000000 \leftarrow 0.500000 \times 2
\end{array}
$$

$$0.828125_{10} = (0.110101)_2$$

1.3.2 Algoritmos generales de conversión

Los ejemplos presentados hasta ahora ilustran los principios de la conversión de bases. Con frecuencia, es útil definir procedimientos generales para resolver diversos problemas de modo que se puedan aplicar los pasos básicos en el orden adecuado. Ahora formularemos los métodos de conversión de base que utilizamos como dos algoritmos de conversión generalizados.

Algoritmo 1.1

Para convertir un número N de base A a base B, utilizamos

(a) el método de sustitución de una serie con aritmética de base B, o

(b) el método de división entre la base o multiplicación por la base con aritmética de base A.

Podemos utilizar el algoritmo 1.1 para la conversión entre dos bases cualesquiera. Sin embargo, al hacer esto, tal vez sea necesario utilizar la aritmética en una base poco familiar. El siguiente algoritmo supera esta dificultad a costa de un procedimiento más largo.

Algoritmo 1.2

Para convertir un número N de base A a base B, utilizamos

(a) el método de sustitución de una serie con aritmética de base 10 para convertir N de base A a base 10, y

(b) el método de división entre la base o multiplicación por la base con aritmética decimal, para convertir N de base 10 a base B.

En general, el algoritmo 1.2 requiere más pasos que el algoritmo 1.1, Pero, este último es más sencillo, rápido, y menos propenso a errores, ya que toda la aritmética se realiza en forma decimal.

EJEMPLO 1.23

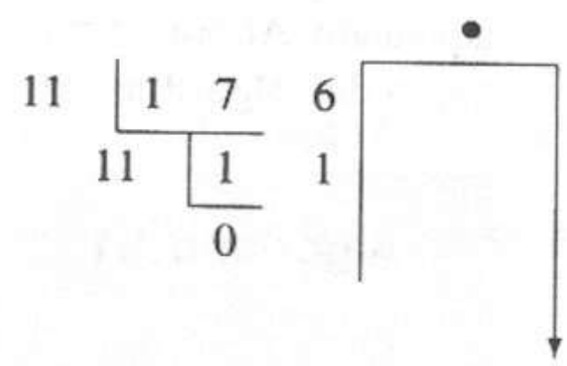
Convertir $(18.6)_9 = (?)_{11}$

$$NA = (18.6)9$$

a. Al convertir a base 10 por sustitución de una serie obtenemos

$$\begin{aligned} N_{10} &= 1 \times 9^1 + 8 \times 9^0 + 6 \times 9^{-1} \\ &= 9 + 8 + 0.666... \\ &= (17.666...)_{10} \end{aligned}$$

b. Al convertir de base 10 a base 11 por división entre la base obtenemos



$7.326 \leftarrow 0.666 \times 11$
$3.586 \leftarrow 0.326 \times 11$
$6.446 \leftarrow 0.586 \times 11$

Al unir las partes entera y fraccionaria,

$$N_{11} = (16.736...)_{11}.$$

1.3.3 Conversión entre la base *A* y la base *B* cuando $B = A^k$

Podemos utilizar procedimientos de conversión más sencillos cuando una base es una potencia de la otra, o sea, $B = A^k$. Los siguientes procedimientos son muy útiles.

Algoritmo 1.3

(a) Para convertir un número N de base A a base B cuando $B = A^k$ y k es un entero positivo, agrupamos los dígitos de N en grupos de k dígitos en ambas direcciones a partir del punto y después remplazamos cada grupo por el dígito equivalente en base B.

(b) Para convertir un número N de base B a base A cuando $B = A^k$ y k es un entero positivo, remplazamos cada dígito en base B de N por los k dígitos equivalentes en base A.

Los siguientes ejemplos ilustran la potencia y velocidad de este algoritmo para el caso $A = 2$.

EJEMPLO 1.24

Convertir $(1011011.1010111)_2$ a base 8.

Podemos aplicar el algoritmo 1.3a, con $B = 8 = 2^3 = A^k$. Por tanto, agrupamos tres dígitos binarios por cada dígito octal.

$$\underbrace{001}_{1}\ \underbrace{011}_{3}\ \underbrace{011}_{3}\ .\ \underbrace{101}_{5}\ \underbrace{011}_{3}\ \underbrace{100}_{4}$$

$$1011011.1010111_2 = (133.534)_8$$

EJEMPLO 1.25

Convertir $(AF.16C)_{16}$ a base 8.

Como 16 y 8 son potencias de 2, podemos aplicar el algoritmo 1.3 dos veces, como sigue.

- Usamos el algoritmo 1.3b para convertir $(AF.16C)16$ a base 2, ya que $16 = 2^4$. Remplazamos cada dígito hexadecimal por cuatro dígitos binarios.

$$\overbrace{1010}^{A}\ \overbrace{1111}^{F}\ .\ \overbrace{0001}^{1}\ \overbrace{0110}^{6}\ \overbrace{1100}^{C}$$

$$(AF.16C)_{16} = (10101111.0001011011)_2$$

- Usamos el algoritmo 1.3a para convertir el número binario a base 8.

$$\underbrace{010}_{2}\ \underbrace{101}_{5}\ \underbrace{111}_{7}\ .\ \underbrace{000}_{0}\ \underbrace{101}_{5}\ \underbrace{101}_{5}\ \underbrace{100}_{4}$$

Por tanto,

$$(AF.16C)_{16} = (257.0554)_8$$

1.4 Representación de números con signo

El signo de los números almacenados en los sistemas digitales se especifica mediante un dígito llamado *dígito de signo*, que por lo general se coloca en la posición de extrema izquierda de los dígitos del número, como se ilustra en la figura 1.2. Los números positivos se especifican con un dígito de signo igual a cero, y los negativos, con un dígito de signo distinto de cero. La magnitud de un número positivo se representa simplemente mediante sus dígitos posicionales, pero existen varios métodos para representar la magnitud de los números negativos. La tabla 1.6 ilustra los métodos de magnitud y signo, de complemento a la base y de complemento disminuido a la base. A continuación analizaremos cada método con más detalle.

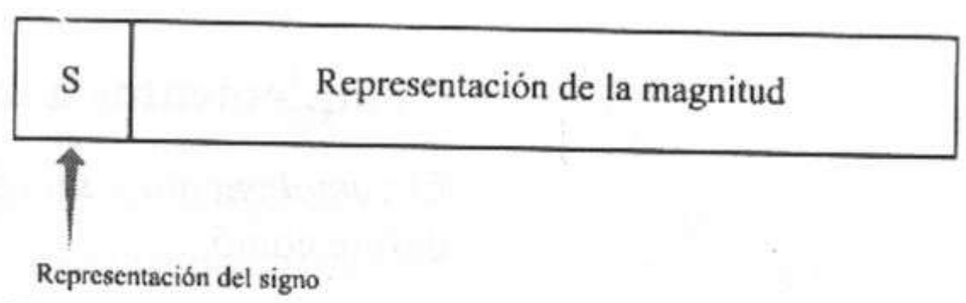


Figura 1.2 Formato de un número con signo

1.4.1 Números con magnitud y signo

El método más sencillo para representar los números con signo es el de *magnitud y signo*. Sin embargo, el empleo de este método requiere circuitos aritméticos y algoritmos con un costo mayor en términos de componentes y de tiempo de cálculo, en relación con otros métodos; por tanto, no suele utilizarse en la práctica para representar números enteros.

Podemos escribir un número con signo $N = \pm(a_{n-1}...a_0.a_{-1}...a_{-m})_r$ en el formato de magnitud y signo como sigue.

$$N = (sa_{n-1}...a_0.a_{-1}...a_{-m})_{rsm} \tag{1.6}$$

donde $s = 0$ si N es positivo y $s = r - 1$ si N es negativo.

EJEMPLO 1.26

Determinar el código de magnitud y signo de $N = -(13)_{10}$ en binario ($r = 2$) y decimal ($r = 10$).

En binario:

$$\begin{aligned} N &= -(13)_{10} \\ &= -(1101)_2 \\ &= (1,1101)_{2sm} \end{aligned}$$

En decimal:

$$\begin{aligned} N &= -(13)_{10} \\ &= (9, 13)_{10sm} \end{aligned}$$

donde 9 representa el signo negativo para $r = 10$.

En la tabla 1.6 se dan más ejemplos de números binarios con magnitud y signo. Para mayor claridad, utilizamos las comas para delimitar los dígitos de signo.

1.4.2 Sistemas numéricos complementarios

Los números complementarios son la base de la aritmética complementaria, un método de gran utilidad que se emplea con frecuencia en los sistemas digitales para realizar operaciones matemáticas con números con signo. En estos sistemas, los números positivos se representan de la misma manera que en un sistema con magnitud y signo, mientras que los números negativos se representan como el complemento del número positivo correspondiente. El complemento a una base y el complemento disminuido a una base son sistemas numéricos importantes que analizaremos a continuación. La aritmética complementaria se ilustra también con ejemplos en este capítulo.

Complementos a una base

El *complemento a una base* $[N]_r$ de un número $(N)_r$ dado por la ecuación 1.2 se define como

$$[N]_r = r^n - (N)_r \tag{1.7}$$

TABLA 1.6 EJEMPLOS DE REPRESENTACIÓN DE NÚMEROS CON SIGNO*

Signo decimal	Binario en magnitud y signo	Sistema de complemento a dos	Sistema de complemento a uno
+15	0,1111	0,1111	0,1111
+14	0,1110	0,1110	0,1110
+13	0,1101	0,1101	0,1101
+12	0,1100	0,1100	0,1100
+11	0,1011	0,1011	0,1011
+10	0,1010	0,1010	0,1010
+9	0,1001	0,1001	0,1001
+8	0,1000	0,1000	0,1000
+7	0,0111	0,0111	0,0111
+6	0,0110	0,0110	0,0110
+5	0,0101	0,0101	0,0101
+4	0,0100	0,0100	0,0100
+3	0,0011	0,0011	0,0011
+2	0,0010	0,0010	0,0010
+1	0,0001	0,0001	0,0001
0	0,0000	0,0000	0,0000
	(1,0000)		(1,1111)
-1	1,0001	1,1111	1,1110
-2	1,0010	1,1110	1,1101
-3	1,0011	1,1101	1,1100
-4	1,0100	1,1100	1,1011
-5	1,0101	1,1011	1,1010
-6	1,0110	1,1010	1,1001
-7	1,0111	1,1001	1,1000
-8	1,1000	1,1000	1,0111
-9	1,1001	1,0111	1,0110
-10	1,1010	1,0110	1,0101
-11	1,1011	1,0101	1,0100
-12	1,1100	1,0100	1,0011
-13	1,1101	1,0011	1,0010
-14	1,1110	1,0010	1,0001
-15	1,1111	1,0001	1,0000
-16	—	1,0000	—

* Note que los bits de signo están delimitados por comas.

donde n es el número de dígitos de $(N)_r$. El número positivo más grande (llamado *escala total positiva*) que puede representarse es $r^{n-1} - 1$, mientras que el número negativo más pequeño (llamado *escala total negativa*) es $-r^{n-1}$.

El *complemento a dos* es un caso especial del complemento a una base para números binarios ($r = 2$) y está dado por

$$[N]_2 = 2^n - (N)_2 \tag{1.8}$$

donde n es el número de bits de $(N)_2$. El complemento a dos es el formato de uso más común para los números con signo en los sistemas digitales y, por tanto, será el centro de la mayor parte de los ejemplos de este texto.

Los siguientes ejemplos ilustran la forma de determinar el complemento a dos de un número binario utilizando la ecuación 1.8.

Ejemplo 1.27

Determinar el complemento a dos de $(N)_2 = (01100101)_2$.

De la ecuación 1.8,

$$\begin{aligned}[N]_2 &= [01100101]_2 \\ &= 2^8 - (01100101)_2 \\ &= (100000000)_2 - (01100101)_2 \\ &= (10011011)_2.\end{aligned}$$

Ejemplo 1.28

Determinar el complemento a dos de $(N)_2 = (11010100)_2$ y verificar que puede servir para representar $-(N)_2$, demostrando que $(N)_2 + [N]_2 = 0$.

Primero, determinamos el complemento a dos mediante la ecuación 1.8:

$$\begin{aligned}[N]_2 &= [11010100]_2 \\ &= 2^8 - (11010100)_2 \\ &= (100000000)_2 - (11010100)_2 \\ &= (00101100)_2.\end{aligned}$$

Para verificar que $[N]_2$ puede servir para representar $-(N)_2$, calculemos $(N)_2 + [N]_2$:

$$\begin{array}{cccccccccc} & & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ & + & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & \uparrow & & & & & & & & \\ & \text{acarreo} & & & & & & & & \end{array}$$

Si descartamos el bit de acarreo, obtenemos $(N)_2 + [N]_2 = (00000000)_2$; es decir, la suma de un número binario y su complemento a dos es 0. Por tanto, podemos utilizar $[N]_2$ para representar $-(N)_2$.

Ejemplo 1.29

Determinar el complemento a dos de $[N]_2 = (00101100)_2$, calculado en el ejemplo 1.28.

$$[[N]_2]_2 = [00101100]_2$$

$$= 2^8 - (00101100)_2$$
$$= (100000000)_2 - (00101100)_2$$
$$= (11010100)_2$$

Observe que el resultado es el valor original de $(N)_2$ dado en el ejemplo 1.28.

Del último ejemplo, vemos que si aplicamos dos veces la operación de complemento a dos a un número, obtenemos su valor original. Podemos verificar esto con facilidad para el caso general si sustituimos $-(N)_2$ por $[N]_2$, como sigue:

$$[[N]_2]_2 = [-(N)_2]_2$$
$$= -(-(N)_2)_2$$
$$= (N)_2$$

EJEMPLO 1.30

Determinar el complemento a dos de $(N)_2 = (10110)_2$ para $n = 8$.

De la ecuación 1.8,

$$[N]_2 = [10110]_2$$
$$= 2^8 - (10110)_2$$
$$= (100000000)_2 - (10110)_2$$
$$= (11101010)_2$$

Observe que mantenemos 8 bits en el resultado. Aconsejamos al lector verificar el hecho de que este valor de $[N]_2$ puede servir para representar $-(N)_2$ y que $[[N]_2]_2 = (N)_2$.

El siguiente ejemplo muestra que el procedimiento básico para determinar el complemento a una base de un número es el mismo para cualquier base.

EJEMPLO 1.31

Determinar el complemento a 10 de $(N)_{10} = (40960)_{10}$.

De la ecuación 1.7,

$$[N]_{10} = [40960]_{10}$$
$$= 10^5 - (40960)_{10}$$
$$= (100000)_{10} - (40960)_{10}$$
$$= (59040)_{10}.$$

Observe que mantenemos 5 dígitos en el resultado. El lector debe verificar que $[N]_{10}$ puede servir para representar $-(N)_{10}$ y que $[[N]_{10}]_{10} = (N)_{10}$.

Aunque siempre podemos determinar el complemento a una base de un número mediante la definición de la ecuación 1.7, hay métodos más sencillos. Presentamos los siguientes dos algoritmos para calcular $[N]_r$, dado $(N)_r$, sin demostración.

Algoritmo 1.4 Determinar $[N]_r$ dado $(N)_r$.

Copiamos los dígitos de N, a partir del menos significativo y procediendo hacia el más significativo, hasta llegar al primer dígito distinto de cero. Remplazamos este dígito, a_i, con $r - a_i$. Después, en caso necesario, continuamos remplazando cada dígito restante aj de N por $(r - 1) - a_j$ hasta haber remplazado el dígito más significativo.

Para el caso especial de los números binarios ($r = 2$), el primer dígito distinto de cero, a_i, es siempre 1. Por tanto, remplazamos a_i por $r - a_i = 2 - 1 = 1$, de modo que ai permanece sin cambio. Cada bit restante a_j se remplaza por $(r - 1) - a_j = 1 - a_j = \bar{a}_j$. Por tanto, la aplicación del algoritmo 1.4 a los números binarios sólo copia todos los bits hasta el primer bit igual a 1 inclusive, y complementa después los bits restantes.

EJEMPLO 1.32

Determinar el complemento a dos de $N = (01100101)_2$.

$$N = 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1$$
$$[N]_2 = (1\ 0\ 0\ 1\ 1\ 0\ 1\ 1)_2$$

primer dígito distinto de cero

EJEMPLO 1.33

Determinar el complemento a dos de $N = (11010100)_2$.

$$N = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0$$
$$[N]_2 = (0\ 0\ 1\ 0\ 1\ 1\ 0\ 0)_2$$

primer dígito distinto de cero

EJEMPLO 1.34

Determinar el complemento a dos de $N = (10110)_2$ para $n = 8$.

En primer lugar, como $n = 8$, debemos concatenar tres ceros en las posiciones de los bits más significativos para formar un número de 8 bits. Después aplicamos el algoritmo 1.4.

$$N = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$$
$$[N]_2 = (1\ 1\ 1\ 0\ 1\ 0\ 1\ 0)_2$$

primer dígito distinto de cero

EJEMPLO 1.35

Determinar el complemento a diez de $(40960)_{10}$.

$$N = 4\ 0\ 9\ 6\ 0$$
$$[N]_{10} = (5\ 9\ 0\ 4\ 0)_{10}$$

primer dígito distinto de cero

Algoritmo 1.5 Determinar $[N]_r$ dado $(N)_r$.

Primero remplazamos cada dígito, a_k, de $(N)_r$ por $(r - 1) - a_k$ y después sumamos 1 al resultado.

En el caso particular de los números binarios ($r = 2$), remplazamos cada bit a_k, por $(r - 1) - a_k = 1 - a_k = \overline{a}_k$. Por tanto, aplicamos el algoritmo 1.5 complementando cada bit y sumando 1 al resultado.

EJEMPLO 1.36

Determinar el complemento a dos de $N = (01100101)_2$.

$$\begin{aligned} N = \; & 01100101 \\ & 10011010 \quad \text{Complementar los bits} \\ & \quad\quad +1 \quad \text{Sumar 1} \\ [N]_2 = \; & (10011011)_2 \end{aligned}$$

EJEMPLO 1.37

Determinar el complemento a dos de $N = (11010100)_2$.

$$\begin{aligned} N = \; & 11010100 \\ & 00101011 \quad \text{Complementar los bits} \\ & \quad\quad +1 \quad \text{Sumar 1} \\ [N]_2 = \; & (00101100)_2 \end{aligned}$$

EJEMPLO 1.38

Determinar el complemento a diez de $(40960)_{10}$.

$$\begin{aligned} N = \; & 40960 \\ & 59039 \quad \text{Complementar los dígitos} \\ & \quad\quad +1 \quad \text{Sumar 1} \\ [N]_{10} = \; & (59040)_{10} \end{aligned}$$

Observe que el algoritmo 1.4 es adecuado para los cálculos a mano, mientras que el 1.5 es más útil para implantación en máquina, pues no implica toma de decisiones.

Sistemas numéricos de complemento a una base

Hemos definido el complemento a una base, y presentado e ilustrado varios métodos para determinar el complemento a una base de un número dado. También sugerimos, por ejemplo, que podemos utilizar el complemento a una base de un número dado para representar el negativo de ese número. Ahora describiremos con más precisión un sistema numérico que utiliza el complemento a dos para representar los números negativos. Podemos definir de manera análoga sistemas para otras bases.

En el *sistema numérico de complemento a dos*, los valores positivos se representan de la misma forma que en el sistema de magnitud y signo; utilizamos un bit de 0 a la izquierda para representar el signo. Los números negativos se representan con el complemento a dos del número positivo correspondiente. Utilizaremos la notación $(N)_{2cns}$ para denotar un número representado en el sistema numérico de complemento a dos. Así, $N = +(a_{n-2},\ldots,a_0)_2 = (0,a_{n-2},\ldots,a_0)_{2cns}$, donde $0 \le N \le 2^{n-1} - 1$. Si $N = (a_{n-1},a_{n-2},\ldots,a_0)_2$, entonces $-N$ se representa en el sistema numérico de complemento a dos como $[a_{n-1},\ldots,a_0]_2$, donde $-1 \ge -N \ge -2^{n-1}$. Todos los números negativos en el sistema numérico de complemento a dos tienen un bit de signo igual a 1. La tabla 1.6 muestra los códigos del sistema numérico de complemento a dos para $n = 5$.

Los siguientes ejemplos ilustran la codificación de los números positivos y negativos en el sistema numérico de complemento a dos. Se aconseja al lector verificar los datos de complemento a dos de la tabla 1.6 después de estudiar los ejemplos.

EJEMPLO 1.39

Dado $(N)_2 = (1100101)_2$, determinar las representaciones de $\pm(N)_2$ en el sistema numérico de complemento a dos, para $n = 8$.

Por inspección,

$$+(N)_2 = (0, 1100101)_{2cns}$$

De la ecuación 1.8,

$$\begin{aligned} -(N)_2 &= [+(N)_2]_2 \\ &= [0, 1100101]_2 \\ &= 2^8 - (0, 1100101)_2 \\ &= (100000000)_2 - (0, 1100101)_2 \\ &= (1, 0011011)_{2cns}. \end{aligned}$$

Por sus bits de signo, vemos que $(0,1100101)_{2cns}$ representa un valor positivo y $(1,0011011)_{2cns}$ es su negativo. En este ejemplo y los posteriores, utilizaremos una coma para facilitar la identificación del bit de signo.

EJEMPLO 1.40

Determinar las representaciones en el sistema numérico de complemento a dos de $\pm(110101)_2$ para $n = 8$.

Por inspección,

$$+(110101)_2 = (0, 0110101)_{2cns}$$

De la ecuación 1.8,

$$\begin{aligned} -(110101)_2 &= [110101]_2 \\ &= 2^8 - (110101)_2 \\ &= (100000000)_2 - (110101)_2 \\ &= (1, 1001011)_{2cns} \end{aligned}$$

EJEMPLO 1.41

Determinar la codificación en el sistema numérico de complemento a dos de $-(13)_{10}$ para $n = 8$.

Comenzamos con la conversión de $(13)_{10}$ de decimal a binario.

$$+(13)_{10} = +(1101)_2 = (0,0001101)_{2cns}$$

Después calculamos el complemento a dos de $(0,0001101)_{2cns}$ para representar $-(13)_{10}$:

$$\begin{aligned} -(13)_{10} &= -(0, 0001101)_{2cns} \\ &= [0, 0001101]_2 \\ &= 2^8 - (0, 0001101)_2 \\ &= (1, 1110011)_{2cns} \end{aligned}$$

EJEMPLO 1.42

Determinar el número decimal representado por $N = (1,1111010)_{2cns}$.

Por bit de signo, vemos que N es un número negativo. Así, determinamos la magnitud de N (el valor positivo correspondiente) calculando su complemento a dos.

$$\begin{aligned} N &= (1,1111010)_{2cns} \\ &= -[1,1111010]_2 \\ &= -(2^8 - (1,1111010)_2) \\ &= -(0,0000110)_{2cns} \\ &= -(6)_{10} \end{aligned}$$

donde $(0,0000110)_{2cns} = +(6)_{10}$. Por tanto, $(1,1111010)_{2cns}$ representa a $-(6)_{10}$.

Consideremos ahora algunos ejemplos de la aritmética con números de complemento a una base.

Aritmética de complemento a una base

Muchas computadoras digitales utilizan un sistema numérico de complemento a una base a fin de minimizar la cantidad de circuitos necesarios para realizar la aritmética de enteros. Por ejemplo, podemos realizar la operación $A - B$ calculando $A + (-B)$, donde $(-B)$ está representado por el complemento a dos de B. Por tanto, la computadora sólo necesita un sumador binario y algunos circuitos complementarios para la suma y la resta. Este punto de vista es conveniente para el análisis de la aritmética de complemento a una base y por ello lo utilizaremos en los párrafos siguientes. Como en una computadora la aritmética se realiza principalmente en binario, centraremos nuestro análisis en la aritmética de complemento a dos.

Antes de iniciar un análisis profundo, consideremos una limitación fundamental de la representación de números en la máquina. Las máquinas del tipo de las computadoras digitales operan con sistemas de números finitos impuestos por el número de bits que se pueden utilizar en la representación de cantidades numéricas. El número de bits disponibles en la unidad aritmética de la computadora limita el intervalo numérico que se puede representar en la máquina. El sistema no puede manejar números fuera de este intervalo. Las máquinas que utilizan el sistema numérico de complemento a dos (2cns) pueden representar enteros en el intervalo

$$-2^{n-1} \leq N \leq 2^{n-1} - 1 \tag{1.9}$$

donde n es el número de bits disponibles para representar N. Observe que $2^{n-1} - 1 = (0,11...1)_{2cns}$ y que $-2^{n-1} = (1,00...0)_{2cns}$ (el bit del extremo izquierdo representa el signo y los $n - 1$ bits restantes representan la magnitud).

Si una operación produce un resultado fuera del intervalo disponible definido en la ecuación 1.9; es decir, si $N > 2^{n-1} - 1$ o $N < -2^{n-1}$, decimos que se presenta una *condición de desbordamiento*. En estos casos, el número de n bits originado por la operación no será una representación válida del resultado. Las computadoras digitales vigilan sus operaciones aritméticas al realizar aritmética

de complemento a dos y generan una señal de advertencia cuando se presenta un desbordamiento, a fin de que los números no válidos no sean considerados como resultados correctos.

Ahora consideraremos tres casos para ilustrar la aritmética en el sistema numérico de complemento a dos: $A = B + C$, $A = B - C$ y $A = -B - C$. Describiremos cada caso en general y después lo aclararemos con ejemplos apropiados. Para todos los casos, suponga que $B \geq 0$ y $C \geq 0$. Los resultados se pueden generalizar fácilmente para incluir valores negativos de B y C.

Caso 1. Calcular $A = B + C$. Como B y C son no negativos, A también será no negativo, y esto simplemente se convierte en

$$(A)_2 = (B)_2 + (C)_2$$

Como los tres números son positivos, no tenemos que utilizar el complemento a dos.

La única dificultad que puede surgir en este caso es cuando $A > 2^{n-1} - 1$, es decir, cuando se presenta un desbordamiento. Es fácil detectar tal condición, ya que el bit de signo de A será incorrecto. Para mostrar esto, consideremos la suma de los dos máximos números positivos de n bits representables.

$$0 \leq A \leq (2^{n-1} - 1) + (2^{n-1} - 1) = 2^n - 2$$

Como el máximo valor positivo de n bits representable es $2^{n-1} - 1$, se presenta una condición de desbordamiento para cualquier suma en el intervalo

$$A \geq 2^{n-1}$$

El $n^{\text{-ésimo}}$ bit de un número binario en este intervalo adoptará el valor 1. Por desgracia, éste es el bit que representa el signo en un número de n bits de complemento a dos. Por tanto, el resultado parece ser negativo, lo que indica la condición de desbordamiento.

Debe observarse que si $A < 2^n$, nunca habrá un acarreo más allá del $n^{\text{-ésimo}}$ bit del sumador binario.

Los siguientes ejemplos utilizarán el sistema numérico de complemento a dos de 5 bits, cuyos valores se dan en la tabla 1.6.

EJEMPLO 1.43

Calcular $(9)_{10} + (5)_{10}$ con aritmética de complemento a dos de 5 bits.

Comenzamos escribiendo $(9)_{10}$ y $(5)_{10}$ como números de complemento a dos de 5 bits. Puesto que ambos números son positivos, utilizamos un bit de signo 0 para cada uno. De la tabla 1.6,

$$+(9)_{10} = +(1001)_2 = (0, 1001)_{2cns}$$
$$+(5)_{10} = +(0101)_2 = (0, 0101)_{2cns}$$

Al sumar estos dos códigos de 5 bits obtenemos

$$\begin{array}{cccccc} & 0 & 1 & 0 & 0 & 1 \\ + & 0 & 0 & 1 & 0 & 1 \\ \hline & 0 & 1 & 1 & 1 & 0 \end{array}$$

Como el resultado también tiene un bit de signo 0, representa correctamente la suma positiva deseada, que se interpreta como

$$(0, 1110)_{2cns} = +(1110)_2 = +(14)_{10}$$

EJEMPLO 1.44

Calcular $(12)_{10} + (7)_{10}$.

De la tabla 1.6,

$$+(12)_{10} = +(1100)_2 = (0,1100)_{2cns}$$
$$+(7)_{10} = +(0111)_2 = (0,0111)_{2cns}$$

Sumando los dos códigos de 5 bits resulta

$$\begin{array}{cccccc} & 0 & 1 & 1 & 0 & 0 \\ + & 0 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 1 & 1 \end{array}$$

El resultado es $(1,0011)_{2cns}$, que se interpreta, según la tabla 1.6, como

$$(1,0011)_{2cns} = -(1101)_2 = -(13)_{10}$$

Si examinamos más de cerca este cálculo vemos que ¡la suma de dos números positivos parece haber dado un resultado negativo! Sin embargo, esto no puede ser correcto, por lo que debe haber una explicación. La respuesta es que la suma de los dos números dados requiere más espacio que los 5 bits asignados para representarla. La suma correcta es $+(19)_{10}$, que está fuera del intervalo de números de complemento a dos de 5 bits, pues la escala total positiva es $(0,1111)_{2cns} = +(15)_{10}$. El bit de signo incorrecto obtenido en los cálculos indica un resultado incorrecto. Por tanto, hay una condición de desbordamiento.

Caso 2. Calcular $A = B - C$. Consideramos este cálculo como $A = B + (C)$ de la siguiente manera. Queremos calcular

$$A = (B)_2 + (-(C)_2)$$

Suponga que representamos esta operación codificando los números en complemento a dos. El número positivo $(B)_2$ no cambia, pero $-(C)_2$ se convierte en $[C]_2$:

$$\begin{aligned} A &= (B)_2 + [C]_2 \\ &= (B)_2 + 2^n - (C)_2 \\ &= 2^n + (B - C)_2 \end{aligned}$$

Por tanto, el cálculo equivale a $2^n + (B - C)$. Ésta es la respuesta deseada, excepto que aparece un término 2^n adicional. ¿Podemos ignorarlo? Si $B \geq C$, entonces $B - C \geq 0$, lo que hace $A \geq 2^n$. El término 2^n representa un bit de acarreo y se puede descartar, conservando $(B - C)_2$ (un sumador binario de n bits generará un acarreo para cualquier suma $A \geq 2^n$). Por tanto,

$$(A)_2 = (B)_2 + [C]_2 \,|\, \text{ acarreo descartado}$$

Si $B < C$, entonces $B - C < 0$, lo que da $A = 2^n - (C - B)_2 = [C - B]_2$, o $A = -(C - B)_2$ que es la respuesta deseada. Observe que en este caso no hay acarreo. Más adelante resumiremos todas las condiciones posibles en forma tabular.

Si B y C son ambos números positivos, la magnitud de $B - C$ siempre será menor que cualquiera de los dos números. Esto significa que no se puede presentar un desbordamiento al calcular $B - C$.

EJEMPLO 1.45

Calcular $(12)_{10} - (5)_{10}$.

Realizamos este cálculo como $(12)_{10} + (-(5)_{10})$

$$(12)_{10} = +(1100)_2 = (0, 1100)_{2cns}$$
$$-(5)_{10} = -(0101)_2 = (1, 1011)_{2cns}$$

Sumamos los dos códigos de 5 bits para obtener

$$\begin{array}{cccccc} & 0 & 1 & 1 & 0 & 0 \\ + & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

↑
Acarreo

Al descartar el acarreo, el bit de signo es 0 y, por tanto, interpretamos el resultado como

$$(0, 0111)_{2cns} = +(0111)_2 = +(7)_{10}$$

EJEMPLO 1.46

Invertir el orden de los operandos del ejemplo anterior, y calcular $(5)_{10} - (12)_{10}$.

Realizamos el cálculo como $(5)_{10} + (-(12)_{10})$.

$$(5)_{10} = +(0101)_2 = (0, 0101)_{2cns}$$
$$-(12)_{10} = -(1100)_2 = (1, 0100)_{2cns}$$

Sumamos los dos códigos de 5 bits para obtener

$$\begin{array}{cccccc} & 0 & 0 & 1 & 0 & 1 \\ + & 1 & 0 & 1 & 0 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 \end{array}$$

En este caso no hay acarreo y el bit de signo es 1, lo que indica el siguiente resultado negativo:

$$(1, 1001)_{2cns} = -(0111)_2 = -(7)_{10}$$

EJEMPLO 1.47

Calcular $(0, 0111)_{2cns} - (1, 1010)_{2cns}$.

Realizamos los cálculos como $(0, 0111)_{2cns} + (-(1, 1010)_{2cns})$. El operando del lado izquierdo ya tiene el formato del sistema numérico de complemento a dos. Como su bit de signo es 1, el operando del lado derecho representa un número negativo. Para negarlo, calculamos el complemento a dos de este número negativo para obtener el valor positivo correspondiente. Observe que según la definición de complemento a dos es

$$\begin{aligned} -[A]_2 &= [[A]_2]_2 \\ &= 2^n - [A]_2 \\ &= 2^n - (2^n - A) \\ &= A \end{aligned}$$

Por tanto,

$$-(1, 1010)_{2cns} = (0, 0110)_{2cns}$$

Al sumar los dos códigos de 5 bits obtenemos

$$\begin{array}{rccccc} & 0 & 0 & 1 & 1 & 1 \\ + & 0 & 0 & 1 & 1 & 0 \\ \hline & 0 & 1 & 1 & 0 & 1 \end{array}$$

El resultado es positivo, como lo indica el bit de signo 0, y lo interpretamos como

$$(0, 1101)_{2cns} = +(1101)_2 = +(13)_{10}$$

El lector debe verificar que este cálculo equivale a calcular $(7)_{10} - (-(6)_{10}) = (13)_{10}$.

Caso 3. Calcular $A = -B - C$. El resultado deseado es $A = -(B + C) = [B + C]_2$. Representamos $-B$ y $-C$ mediante el complemento a dos de sus magnitudes, y calculamos $A = (-B) + (-C)$. Por tanto,

$$\begin{aligned} A &= [B]_2 + [C]_2 \\ &= 2^n - (B)_2 + 2^n - (C)_2 \\ &= 2^n + 2^n - (B + C)_2 \\ &= 2^n + [B + C]_2 \end{aligned}$$

Si se descarta el bit de acarreo (2^n), el cálculo produce el resultado correcto, la representación en complemento a dos de $-(B + C)_2$.

EJEMPLO 1.48

Calcular $-(9)_{10} - (5)_{10}$.

Realizamos el cálculo como $(-(9)_{10}) + (-(5)_{10})$.

$$-(9)_{10} = -(1001)_2 = (1, 0111)_{2cns}$$
$$-(5)_{10} = -(0101)_2 = (1, 1011)_{2cns}$$

Al sumar los dos códigos de 5 bits obtenemos

$$\begin{array}{rcccccc} & & 1 & 0 & 1 & 1 & 1 \\ & + & 1 & 1 & 0 & 1 & 1 \\ \hline & 1 & 1 & 0 & 0 & 1 & 0 \\ & \uparrow & & & & & \\ & \text{Acarreo} & & & & & \end{array}$$

Al descartar el acarreo tenemos un bit de signo de 1. Por tanto, el resultado es correcto y se interpreta como

$$(1, 0010)_{2cns} = -(1110)_2 = -(14)_{10}$$

Como cuando se suman dos valores positivos, puede ocurrir un desbordamiento al sumar dos números negativos, para dar un resultado en el intervalo

$$A < -2^{n-1}$$

lo que se indica mediante un resultado que tiene su bit de signo incorrecto (es decir, un resultado que parece ser positivo). Ilustraremos esto en el siguiente ejemplo.

EJEMPLO 1.49

Calcular $-(12)_{10} - (5)_{10}$.

Realizamos el cálculo como $(-(12)_{10}) + (-(5)_{10})$

$$-(12)_{10} = -(1100)_2 = (1,0100)_{2cns}$$
$$-(5)_{10} = -(0101)_2 = (1,1011)_{2cns}$$

Al sumar los dos códigos de 5 bits obtenemos

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & 0 \\ + & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 1 & 1 \end{array}$$

↑
Acarreo

Al descartar el acarreo, interpretamos el resultado como

$$(0,1111)_{2cns} = +(1111)_2 = +(15)_{10}$$

Observe que el bit de signo es incorrecto, lo que indica un desbordamiento. Este resultado es "demasiado negativo"; excede en 1 el intervalo numérico en la dirección negativa, pues el resultado deseado era $-(17)_{10}$. En consecuencia, debido al desbordamiento, el resultado se interpreta de manera incorrecta como $+(15)_{10}$.

El siguiente ejemplo muestra la utilidad de la aritmética de complemento a dos en las computadoras digitales.

EJEMPLO 1.50

***A* y *B* son variables enteras de un programa para computadora, con $A = (25)_{10}$ y $B = -(46)_{10}$. Si la computadora utiliza aritmética de complemento a dos con 8 bits, mostrar la forma como calcularía $A + B$, $A - B$, $B - A$ y $-A - B$.**

Las variables A y B se guardarían en la memoria de la computadora con el formato de sistema numérico de 8 bits de complemento a dos:

$$A = +(25)_{10} = (0, 0011001)_{2cns}$$
$$B = -(46)_{10} = -(0, 0101110)_{2cns} = (1, 1010010)_{2cns}$$

En primer lugar, calcularemos los complementos a dos de A y B para representar $-A$ y $-B$, respectivamente:

$$-A = -(25)_{10} = -(0, 0011001)_{2cns} = (1, 1100111)_{2cns}$$
$$-B = -(-(46)_{10}) = -(1, 1010010)_{2cns} = (0, 0101110)_{2cns}$$

Al realizar los cálculos,

$$A + B: \quad \begin{array}{cccccccc} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ + \; 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

El resultado es $(1, 1101011)_{2cns} = -(0, 0010101)_{2cns} = -(21)_{10}$.

$$
\begin{array}{lcccccccc}
 & & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
A - B = A + (-B): & + & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
 & & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

El resultado es $(0, 1000111)_{2cns} = +(71)_{10}$.

$$
\begin{array}{lcccccccc}
 & & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
B - A = B + (-A): & + & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}
$$

El resultado es $(1, 0111001)_{2cns} = -(0, 1000111)_{2cns} = -(71)_{10}$.

$$
\begin{array}{lcccccccc}
 & & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
-A - B = (-A) + (-B): & + & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

El resultado es $(0, 0010101)_{2cns} = +(21)_{10}$. Observe que en los dos últimos casos se descarta el bit de acarreo.

En la tabla 1.7 damos un resumen de la suma y resta de complemento a dos.

TABLA 1.7 RESUMEN DE LA SUMA Y RESTA DE COMPLEMENTO A DOS

Caso*	Acarreo	Bit de signo	Condición	¿Desbordamiento?
$B + C$	0	0	$B + C \leq 2^{n-1} - 1$	No
	0	1	$B + C > 2^{n-1} - 1$	Sí
$B - C$	1	0	$B \leq C$	No
	0	1	$B > C$	No
$-B - C$	1	1	$-(B + C) > -2^{n-1}$	No
	1	0	$-(B + C) < -2^{n-1}$	Sí

* B y C son números positivos.

La aritmética de complemento a una base se puede utilizar con cualquier base no sólo con los números binarios. Para ilustrar este hecho, los siguientes dos ejemplos usarán la aritmética de complemento a 10 con números de tres dígitos.

EJEMPLO 1.51

Sumar $+(75)_{10}$ y $-(21)_{10}$ utilizando aritmética de complemento a 10 con tres dígitos.

En primer lugar, determinamos los códigos de complemento a 10 de los dos números mediante la ecuación 1.7:

$$(75)_{10} = (0, 75)_{10cns}$$
$$-(21)_{10} = (9, 79)_{10cns}$$

Después realizamos el cálculo como $(75)_{10} + (-(21)_{10})$. Al sumar los dos código de 3 dígitos, obtenemos

$$\begin{array}{rrrr} & 0 & 7 & 5 \\ + & 9 & 7 & 9 \\ \hline 1 & 0 & 5 & 4 \end{array}$$

$\uparrow$
Acarreo

Al descartar el dígito de acarreo, el resultado es $(0{,}54)_{10cns} = (54)_{10}$, que es e resultado correcto.

EJEMPLO 1.52

Sumar $+(21)_{10}$ y $-(75)_{10}$.

De nuevo, comenzamos por determinar los códigos de complemento a diez par los dos números, por medio de la ecuación 1.7:

$$(21)_{10} = (0, 21)_{10cns}$$
$$-(75)_{10} = (9, 25)_{10cns}$$

Al sumar los dos códigos de tres dígitos obtenemos

$$\begin{array}{rrrr} & 0 & 2 & 1 \\ + & 9 & 2 & 5 \\ \hline & 9 & 4 & 6 \end{array}$$

El resultado es $(9{,}46)_{10cns}$, donde el 9 indica que este número representa un valor negativo. El lector debe verificar que $(9{,}46)_{10cns}$ es la representación correcta del resultado deseado, $-(54)_{10}$, en un sistema numérico de complemento a diez.

Sistemas numéricos con complemento disminuido a una base

El *complemento disminuido a una base* $[N]_{r-1}$ de un número $(N)_r$ se define como

$$[N]_{r-1} = r^n - (N)_r - 1 \tag{1.10}$$

donde n es el número de dígitos de $(N)_r$.

El *complemento a uno* es un caso particular del complemento disminuido a una base para los números binarios ($r = 2$) y está dado por

$$[N]_{2-1} = 2^n - (N)_2 - 1 \tag{1.11}$$

donde n es el número de bits de $(N)_2$.

Podemos determinar el complemento a uno de un número binario dado en forma directa a partir de la ecuación 1.11, como se muestra en los siguientes ejemplos. El lector debe verificar los datos del complemento a uno de la tabla 1.6 después de estudiar estos ejemplos.

EJEMPLO 1.53

Determinar el complemento a uno de $(01100101)_2$.

De la ecuación 1.11,

$$\begin{aligned}[N]_{2-1} &= 2^8 - (01100101)_2 - 1 \\ &= (100000000)_2 - (01100101)_2 - (00000001)_2 \\ &= (10011011)_2 - (00000001)_2 \\ &= (10011010)_2.\end{aligned}$$

EJEMPLO 1.54

Determinar el complemento a uno de $(11010100)_2$.

De la ecuación 1.11,

$$\begin{aligned}[N]_{2-1} &= 2^8 - (11010100)_2 - (00000001)_2 \\ &= (100000000)_2 - (11010100)_2 - (00000001)_2 \\ &= (00101100)_2 - (00000001)_2 \\ &= (00101011)_2.\end{aligned}$$

EJEMPLO 1.55

Determinar el complemento a nueve de $(40960)_{10}$.

De la ecuación 1.10,

$$\begin{aligned}[N]_{10-1} &= 10^5 - (40960)_{10} - (00001)_{10} \\ &= (100000)_{10} - (40960)_{10} - (00001)_{10} \\ &= (59040)_{10} - (00001)_{10} \\ &= (59039)_{10}.\end{aligned}$$

Aunque siempre podemos determinar el complemento a uno de un número mediante la definición dada en la ecuación 1.11, hay métodos más sencillos. Los ejemplos anteriores sugieren el siguiente algoritmo para el cálculo de $[N]_{r-1}$ dado $(N)_r$, y lo presentamos sin demostración.

Algoritmo 1.6 Determinar $[N]_{r-1}$ dado $(N)_r$

Remplazamos cada dígito a_i de $(N)_r$ por $r - 1 - a_i$. Observe que si $r = 2$, basta con complementar cada bit individual de $(N)_r$.

Al comparar las ecuaciones 1.7 y 1.10, vemos que la relación entre el complemento a una base y el complemento disminuido a una base de un número $(N)r$ es la siguiente.

$$[N]_r = [N]_{r-1} + 1 \tag{1.12}$$

Ahora debe quedar claro que el algoritmo 1.5 para calcular el complemento a una base es consecuencia del algoritmo 1.6.

Podemos formular sistemas numéricos que utilicen el complemento disminuido a una base para representar los números negativos de una manera similar a la que usamos con el complemento a una base. Sin embargo, no haremos esto aquí; nos limitaremos a ilustrar la aritmética.

Aritmética de complemento disminuido a una base

En los siguientes ejemplos ilustramos las principales características de la aritmética de complemento disminuido a una base. Los tres primeros ejemplos se centran en la suma de complemento a uno con diversas combinaciones de operandos positivos y negativos. Los números utilizados en estos ejemplos se tomaron de la tabla 1.6.

EJEMPLO 1.56

Sumar $+(1001)_2$ y $-(0100)_2$.

Representamos el número positivo como 01001 y el negativo con el complemento a uno de 00100, que es 11011. Por tanto, 00100 + 11011 = 100100. Observe que éste *no* es el resultado correcto. Sin embargo, obtenemos el resultado correcto si el acarreo de salida del bit más significativo se suma a la posición de bit menos significativa; es decir, 00100 + 1 = 00101. Este procedimiento se conoce como *acarreo final circular* y es un paso de corrección necesario en la aritmética de complemento disminuido.

EJEMPLO 1.57

Sumar $+(1001)_2$ y $-(1111)_2$.

Representamos el número positivo como 01001 y el negativo como 10000. Esto produce 01001 + 10000 = 11001. Observe que en este caso el acarreo final circular es 0 y, por tanto, no afecta el resultado.

EJEMPLO 1.58

Sumar $-(1001)_2$ y $-(0011)_2$.

Si representamos ambos números con su complemento a uno, obtenemos 10110 + 11100 = 110010. El paso de acarreo final circular produce el resultado correcto, es decir, 10010 + 1 = 10011.

Los siguientes dos ejemplos ilustran la aritmética de complemento a nueve.

EJEMPLO 1.59

Sumar $+(75)_{10}$ y $-(21)_{10}$.

El complemento a nueve de 021 es 978. Por tanto, la operación es 075 + 978 = 1053, que es el resultado correcto después del procedimiento de acarreo final circular: 053 + 1 = 054.

EJEMPLO 1.60

Sumar $+(21)_{10}$ y $-(75)_{10}$.

El cálculo es 021 + 924 = 945, que es el resultado correcto, pues el acarreo final circular es 0.

1.5 Códigos de computadora

Un *código* es un uso sistemático y de preferencia estandarizado de un conjunto dado de símbolos para representar información. En la vida cotidiana aparecen varias formas sencillas de códigos. Por ejemplo, al acercarnos a un semáforo, se sobreentiende que la luz roja significa alto, que la luz verde significa siga y que la señal ámbar significa precaución. En otras palabras, el código es

Luz roja:	Alto
Luz ámbar:	Precaución
Luz verde:	Siga

Otro código conocido es el que se usa en el beisbol. Cuando un *umpire* levanta sus brazos con dos dedos en la mano derecha y tres dedos en la mano izquierda, se sobreentiende que la cuenta del bateador es dos *strikes* y tres bolas. Estos dos sencillos ejemplos ilustran la idea de los códigos y sin duda que el lector podrá imaginar más ejemplos.

Las computadoras y otros sistemas digitales utilizan códigos más complejos para el procesamiento, almacenamiento e intercambio de información de diversos tipos. Tres tipos importantes de códigos para computadora son el numérico, el de caracteres, y el de detección y corrección de errores. A continuación analizaremos brevemente algunos códigos importantes de cada una de estas categorías.

1.5.1 Códigos numéricos

Los códigos numéricos sirven para representar números con fines de procesamiento, y/o de almacenamiento. Los números de punto fijo y de punto flotante son ejemplos de estos códigos.

Números de punto fijo

Los *números de punto fijo* se utilizan para representar ya sea enteros con signo o bien fracciones con signo. En ambos casos se usan los sistemas de magnitud y signo, de complemento a dos o de complemento a uno para representar los valores con signo. Los enteros de punto fijo tienen un punto binario implícito a la derecha del bit menos significativo, como se muestra en la figura 1.3a, y las fracciones de punto fijo tienen el punto binario implícito entre el bit de signo y el bit de magnitud más significativo, como se muestra en la figura 1.3b.

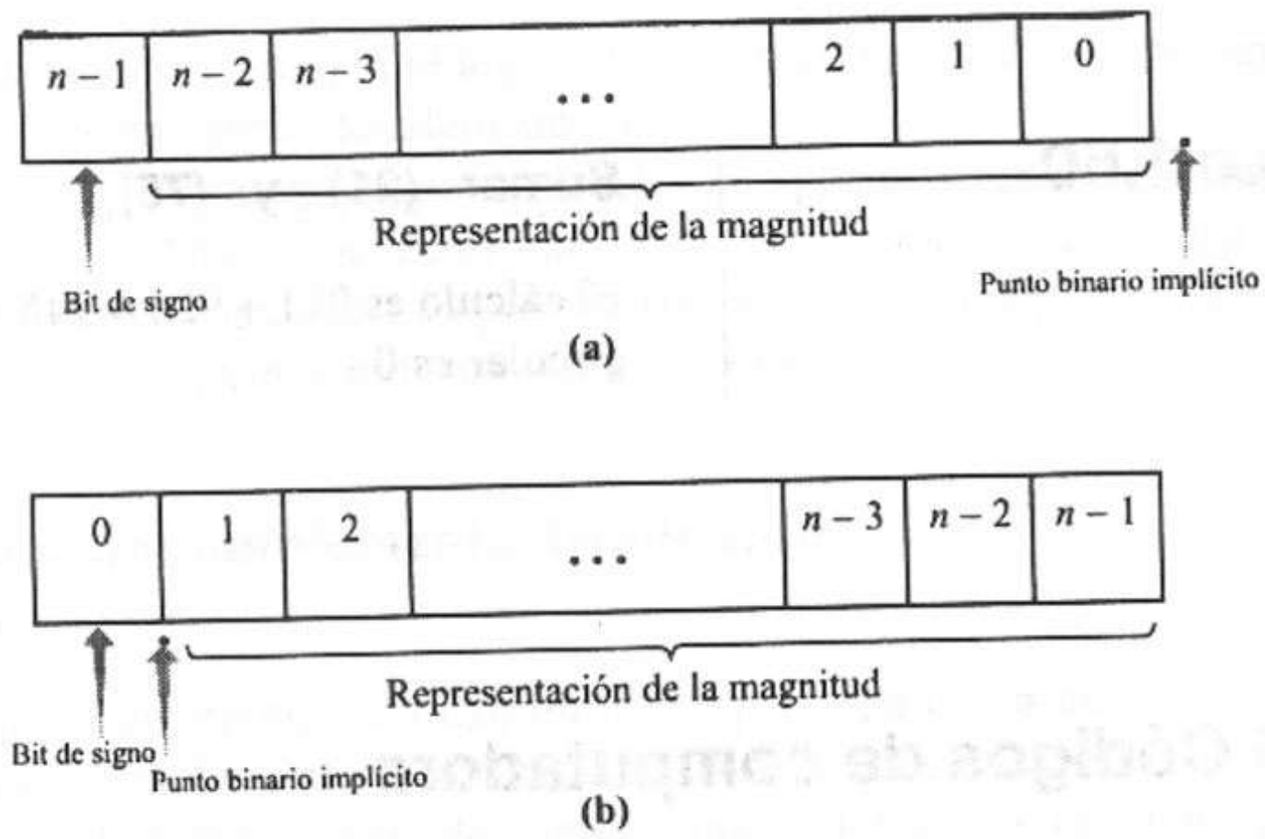


Figura 1.3 Representaciones de números de punto fijo. **(a)** Entero de punto fijo. **(b)** Fracción de punto fijo.

EJEMPLO 1.61

Dar dos posibles interpretaciones del número de punto fijo de 8 bits 01101010, usando el sistema de complemento a dos.

Como el bit de signo es 0, el número representa el entero positivo 1101010. si el punto binario se coloca como en la figura 1.3a, o bien la fracción positiva 0.1101010 si el punto binario se coloca como en la figura 1.3b.

EJEMPLO 1.62

Dar dos posibles interpretaciones del número de punto fijo de 8 bits 11101010, usando el sistema de complemento a dos.

El bit de signo es 1; por tanto, el número representa –0010110. o bien –0.0010110, según sea la convención utilizada para colocar el punto binario.

Representaciones con exceso o sesgadas

Una *representación con exceso –K* de un código *C* se forma sumando el valor *K* a cada palabra de código de *C*. Las representaciones con exceso se utilizan a menudo para representar los exponentes de los números de punto flotante, de modo que el valor mínimo del exponente se represente mediante ceros. Observe que los números con exceso -2^n no son sino los números de complemento a dos, ¡con el bit de signo invertido!

La representación con exceso 8 de la tabla 1.8 se obtiene al sumar $(1000)_2$ al código de 4 bits de complemento a dos. Observe que el resultado es que el número mínimo (–8) se representa con 0000, y el máximo (+7), con 1111.

TABLA 1.8 CÓDIGO CON EXCESO 8

Decimal	Complemento a dos	Exceso 8
+7	0111	1111
+6	0110	1110
+5	0101	1101
+4	0100	1100
+3	0011	1011
+2	0010	1010
+1	0001	1001
0	0000	1000
-1	1111	0111
-2	1110	0110
-3	1101	0101
-4	1100	0100
-5	1011	0011
-6	1010	0010
-7	1001	0001
-8	1000	0000

Números de punto flotante

Los números de punto flotante tienen una forma similar a los números escritos en notación científica. En general, la forma de punto flotante de un número N se escribe como

$$N = M \times r^E \tag{1.13}$$

donde M, la *mantisa* o *significando*, es un número de punto fijo que contiene los dígitos significativos de N y E, el *exponente* o *característica*, es un entero de punto fijo. En el caso general, dado un número de punto fijo N, donde

$$N = \pm(a_{n-1} \ldots a_0.a_{-1} \ldots a_{-m})_r$$

entonces, en forma de punto flotante,

$$N = \pm(.a_{n-1} \ldots a_{-m})_r \times r^n$$

Al deducir la representación de un número de punto flotante, la mantisa y la característica se codifican por separado. La base es implícita y no se incluye en la representación.

La mantisa M muchas veces se codifica con magnitud y signo, por lo general como una fracción, y se puede escribir como

$$M = (S_M.a_{n-1} \ldots a_{-m})_{rsm} \tag{1.14}$$

donde $(.a_{n-1} \ldots a_{-m})_r$ representa la magnitud de M y S_M indica el signo del número. S_M se elige por lo general de modo que

$$M = (-1)^{S_M} \times (.a_{n-1} \ldots a_{-m})_r \tag{1.15}$$

y entonces $S_M = 0$ indica un número positivo, mientras que $S_M = 1$ indica un número negativo.

Con frecuencia, el exponente E se codifica en *complemento a dos con exceso* $-K$. El complemento a dos con exceso $-K$ de un exponente se forma sumando un *sesgo* de K al valor entero en complemento a dos del exponente. En el caso de los números binarios de punto flotante (para los que la base es $r = 2$), K es, por lo general, 2^{e-1}, donde e es el número de bits del exponente. Por tanto,

$$-2^{e-1} \leq E < 2^{e-1}$$
$$0 \leq E + 2^{e-1} < 2^{e}$$

lo que indica que el valor sesgado de E es un número que varía entre 0 y $2^e - 1$ cuando E va de su valor más negativo a su valor más positivo. Podemos escribir la forma con exceso $-K$ de E como

$$E = (b_{e-1}, b_{e-2} \ldots b_0)\text{exceso}-K \tag{1.16}$$

donde b_{e-1} indica el signo de E.

Combinamos M y E, codificados mediante las ecuaciones 1.14 y 1.16, para producir el formato numérico de punto flotante:

$$N = (S_M b_{e-1} b_{e-2} \ldots b_0 a_{n-1} \ldots a_{-m})_r \tag{1.17}$$

lo que representa al número

$$N = (-1)^{S_M} \times (.a_{n-1} \ldots a_{-m})_r \times r^{(b_{e-1} b_{e-2} \ldots b_0)_2 - 2^{e-1}} \tag{1.18}$$

Una excepción al formato de la ecuación 1.17 es el número 0, que se considera como un caso particular y, por lo general, se representa mediante una palabra con ceros.

Las representaciones de punto flotante de un número dado no son únicas. Dado un número N, definido como en la ecuación 1.13, podemos ver que

$$N = M \times r^{E} \tag{1.19}$$
$$= (M \div r) \times r^{E+1} \tag{1.20}$$
$$= (M \times r) \times r^{E-1} \tag{1.21}$$

donde la división $(M \div r)$ se realiza recorriendo los dígitos de M una posición a la derecha, y $(M \times r)$ se realiza recorriendo los dígitos de M una posición a la izquierda. Por tanto, varias combinaciones de mantisa y exponente representan el mismo número. Por ejemplo, sea $M = +(1101.0101)_2$. La representación de M como una fracción con magnitud y signo en el formato de la ecuación 1.14 y la aplicación repetida de la ecuación 1.20 da como resultado

$$M = +(1101.0101)_2$$
$$= (0.11010101)_2 \times 2^4 \tag{1.22}$$
$$= (0.011010101)_2 \times 2^5 \tag{1.23}$$
$$= (0.0011010101)_2 \times 2^6 \tag{1.24}$$
$$\vdots$$

Al realizar cálculos en una computadora, por lo general es más conveniente tener una representación única para cada número. Se utiliza la *normalización* para conferir unicidad a los números de punto flotante. Un número de punto flotante está *normalizado* si el exponente se ajusta de modo que la mantisa tenga un valor distinto de cero en la posición de su dígito más significativo. Por tanto, la ecuación 1.22 es la representación normalizada de N, mientras que los números de las ecuaciones 1.23 y 1.24 no están normalizados.

Observe que el bit más significativo de un número binario normalizado siempre es 1. Por tanto, si representamos M en la forma de magnitud y signo como una fracción normalizada,

$$0.5 \leq |M| < 1.$$

Los formatos de punto flotante que utilizan en los sistemas de cómputo de los diversos fabricantes difieren con frecuencia en la cantidad de bits que se usan para representar la mantisa y el exponente, así como en el método de codificación de cada uno. Casi todos los sistemas utilizan el formato general ilustrado en la figura 1.4, con el signo almacenado en el bit extremo izquierdo, seguido por el exponente y después la mantisa. El formato de una palabra de la figura 1.4a se utiliza por lo general en las computadoras con longitud de palabra de 32 bits o más. El formato de dos palabras de la figura 1.4b se usa en computadoras con longitudes de palabra "cortas" para los números de punto flotante con precisión sencilla o en computadoras con longitudes de palabra largas para la representación con precisión extendida (también llamada doble precisión).

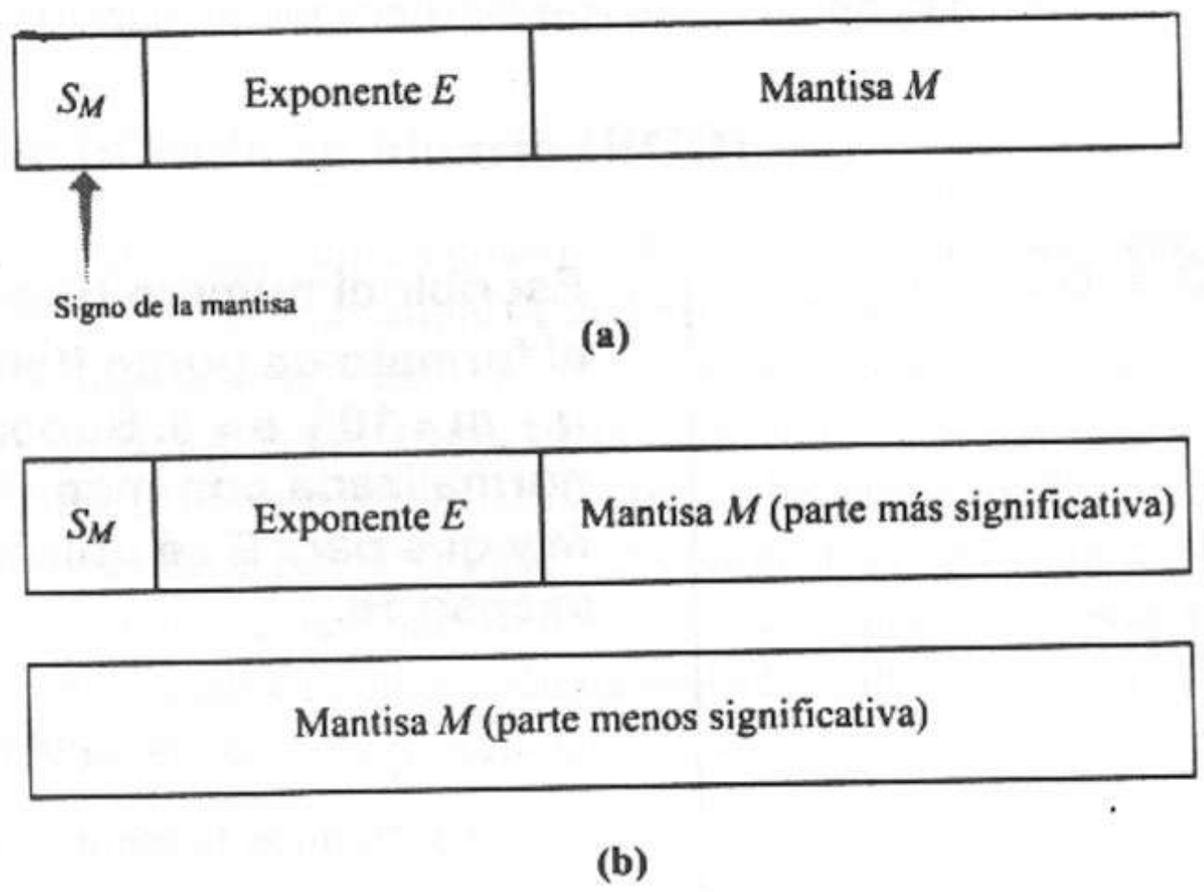


Figura 1.4 Formatos de número de punto flotante. **(a)** Formato típico de precisión simple. **(b)** Formato típico de precisión extendida.

La tabla 1.9 presenta un resumen de los formatos de precisión sencilla y doble utilizados en diversos sistemas de cómputo, incluidos los formatos definidos en el estándar 754-1985 [8] del IEEE. Observe que todos estos formatos emplean un exponente sesgado, con un número variable de bits. Los formatos de la DEC VAX y el estándar de IEEE suprimen el almacenamiento del bit más significativo de la mantisa. Como todos los números son valores binarios en forma normalizada, sabemos que el bit más significativo es 1. Por tanto, no es necesario guardar este bit y se gana un bit adicional para la precisión (denotado por +1 en la tercera columna de la tabla 1.9).

TABLA 1.9 FORMATOS COMUNES PARA NÚMEROS DE PUNTO FLOTANTE*

Sistema/Formato	Total de bits	Bits del significando	Bits del exponente	Sesgo del exponente	Codificación de la mantisa
Estándar 754-1985 del IEEE:					Magnitud/signo (base 2):
Precisión sencilla	32	23(+1)	8	127	$1 \le \lvert M \rvert < 2$
Doble precisión	64	52(+1)	11	1023	$1 \le \lvert M \rvert < 2$
IBM System/360:					Magnitud/signo (base 16):
Precisión sencilla	32	24	7	64	$1/16 \le \lvert M \rvert < 1$
Doble precisión	64	56	7	64	$1/16 \le \lvert M \rvert < 1$
DEC VAX 11/780:					Magnitud/signo (base 2):
Formato F	32	23(+1)	8	128	$1/2 \le \lvert M \rvert < 1$
Formato D	64	55(+1)	8	128	$1/2 \le \lvert M \rvert < 1$
Formato G	64	52(+1)	11	1024	$1/2 \le \lvert M \rvert < 1$
CDC Cyber 70:	60	48	11	1024	Complemento a uno (base 2) $1 \le \lvert M \rvert < 248$

* (+1)⇒ indica que se ha suprimido el bit más significativo.

EJEMPLO 1.63

Escribir el número binario $N = (101101.101)_2$ en el formato de punto flotante de la ecuación 1.17, donde $n + m = 10$ y $e = 5$. Suponga que se utiliza una fracción normalizada con magnitud y signo para representar M y que para E se utiliza el complemento a dos con exceso 16.

$$N = (101101.101)_2 = (0.101101101)_2 \times 2^6$$

Escribimos la mantisa en el formato de la ecuación 1.14:

$$\begin{aligned} M &= +(0.1011011010)_2 \\ &= (0.1011011010)_{2sm} \end{aligned}$$

Codificamos el exponente determinando su forma en complemento a dos y sumando después un sesgo de 16. (Observe que el número de bits del exponente es $e = 5$ y que el valor del sesgo es $2^{e-1} = 2^4 = 16$.) Por tanto,

$$\begin{aligned} E &= +(6)_{10} \\ &= +(0110)_2 \\ &= (00110)_{2cns} \end{aligned}$$

Al sumar el valor de sesgo $16 = (10000)_2$ al complemento a dos de E obtenemos

$$\begin{array}{r} 00110 \\ +\ \ 10000 \\ \hline 10110 \end{array}$$

de modo que

$$E = (1{,}0110)_{\text{exceso}-16}$$

Observe que el signo del exponente, b_{e-1}, es 1, lo cual indica un valor positivo del exponente.

Al unir M y E obtenemos

$$N = (0,\ 1,\ 0110,\ 1011011010)_{fp}$$

Las operaciones aritméticas con números de punto flotante requieren algoritmos especiales para la manipulación de exponentes y mantisas, mismas que rebasan el objetivo de este texto. El lector puede consultar [9] si desea más información acerca de los algoritmos para la aritmética de punto flotante.

1.5.2 Códigos de caracteres y otros códigos

Con frecuencia, es necesario o deseable representar la información como cadenas de caracteres alfabéticos o numéricos. Se han desarrollado muchos códigos de caracteres para esto; ahora analizaremos los más importantes.

Decimal codificado en binario (BCD)

El código *decimal codificado en binario (BCD)* sirve para representar los dígitos decimales del 0 al 9 y es un ejemplo de código *ponderado*, es decir, cada posición de bit en el código tiene un valor o peso numérico fijo asociado a ella. El dígito representado mediante una palabra de código dada se determina sumando los bits ponderados. El código BCD utiliza 4 bits, y los pesos son los mismos que en un entero binario de 4 bits. Por tanto, el código BCD para un dígito decimal dado es igual al equivalente binario del número, con ceros de relleno. Los códigos BCD se conocen también como códigos 8–4–2–1, por los pesos utilizados. El código BCD completo aparece en la tabla 1.10.

TABLA 1.10 CÓDIGO DECIMAL CODIFICADO EN BINARIO (BCD)

0:	0000	5:	0101
1:	0001	6:	0110
2:	0010	7:	0111
3:	0011	8:	1000
4:	0100	9:	1001

Los códigos BCD se utilizan para codificar números que se envían a pantallas numéricas y para representar números en procesadores que realizan aritmética decimal. Esto último se realiza en las computadoras *mainframe* en un extremo del espectro, y en las calculadores de bolsillo en el otro.

EJEMPLO 1.64

Codificar el número decimal $N = (9750)_{10}$ en BCD.

En primer lugar, codificamos los dígitos individuales de la tabla 1.10.

$$9 \rightarrow 1001,\ 7 \rightarrow 0111,\ 5 \rightarrow 0101 \text{ y } 0 \rightarrow 0000$$

Después se concatenan los códigos individuales para obtener

$$N = (1001011101010000)_{BCD}$$

Se han inventado extensiones del código BCD para abarcar no sólo los dígitos decimales sino también los caracteres alfabéticos y otros caracteres que se pueden imprimir, así como caracteres de control que no se imprimen. Estos códigos tienen por lo general una longitud de 6 a 8 bits, y se utilizan para representar datos durante la entrada o salida y para la representación interna de datos no numéricos, como el texto. Uno de estos códigos, utilizado en muchos modelos de computadora *mainframe* de IBM, es el *código de intercambio extendido de decimales codificados en binario* (*EBCDIC*).

ASCII

El código de caracteres más utilizado en las aplicaciones de cómputo es el código *ASCII* (siglas en inglés de Código Estándar Americano para Intercambio de Información), que se pronuncia "aski". El código ASCII de 7 bits está dado en la tabla 1.11. Con frecuencia se utiliza un octavo bit para disponer de la detección de errores. Analizaremos esta técnica, codificación con paridad, en una sección posterior de este capítulo.

EJEMPLO 1.65

Codificar la palabra *Digital* en código ASCII, representando cada caracter con dos dígitos hexadecimales.

Caracter	Código binario	Código hexadecimal
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

Observe que la forma hexadecimal es más compacta y legible que la binaria. Por esta razón, la primera se utiliza con frecuencia para representar información codificada en ASCII.

Códigos Gray

Un *código cíclico* se puede definir como cualquier código en el que, para cualquier palabra de código, un corrimiento circular produce otra palabra del

TABLA 1.11 CÓDIGO DE CARACTERES ASCII

		$c_6c_5c_4$							
		000	001	010	011	100	101	110	111
	0000	NUL	DLE	SP	0	@	P	‘	p
	0001	SOH	DC1	!	1	A	Q	a	q
	0010	STX	DC2	"	2	B	R	b	r
	0011	ETX	DC3	#	3	C	S	c	s
	0100	EOT	DC4	$	4	D	T	d	t
	0101	ENQ	NAK	%	5	E	U	e	u
	0110	ACK	SYN	&	6	F	V	f	v
	0111	BEL	ETB	’	7	G	W	g	w
$c_3c_2c_1c_0$	1000	BS	CAN	(	8	H	X	h	x
	1001	HT	EM	)	9	I	Y	i	y
	1010	LF	SUB	*	;	J	Z	j	z
	1011	VT	ESC	+	;	K	[	k	{
	1100	FF	FS	‘	<	L	\	l	|
	1101	CR	GS	-	=	M	]	m	}
	1110	*S0*	RS	.	>	N	^	n	~
	1111	*S1*	US	/	?	O	_	o	DEL

código. El *código Gray* es uno de los tipos más comunes de códigos cíclicos y tiene la característica de que las palabras de código para dos números consecutivos difieren sólo en un bit. Es decir, la distancia entre las dos palabras de código es 1. En general, la *distancia* entre dos palabras de código binario es igual al número de bits en que difieren las dos palabras.

EJEMPLO 1.66

Definir un código Gray para codificar los números decimales de 0 a 15.

Se necesitan cuatro bits para representar todos los números, y podemos construir el código necesario asignando al bit i de la palabra de código el valor 0 si los bits i e $i + 1$ del número binario correspondiente son iguales, y 1 en caso contrario. El bit más significativo del número siempre se debe comparar con 0 al utilizar esta técnica. El código resultante aparece en la tabla 1.12.

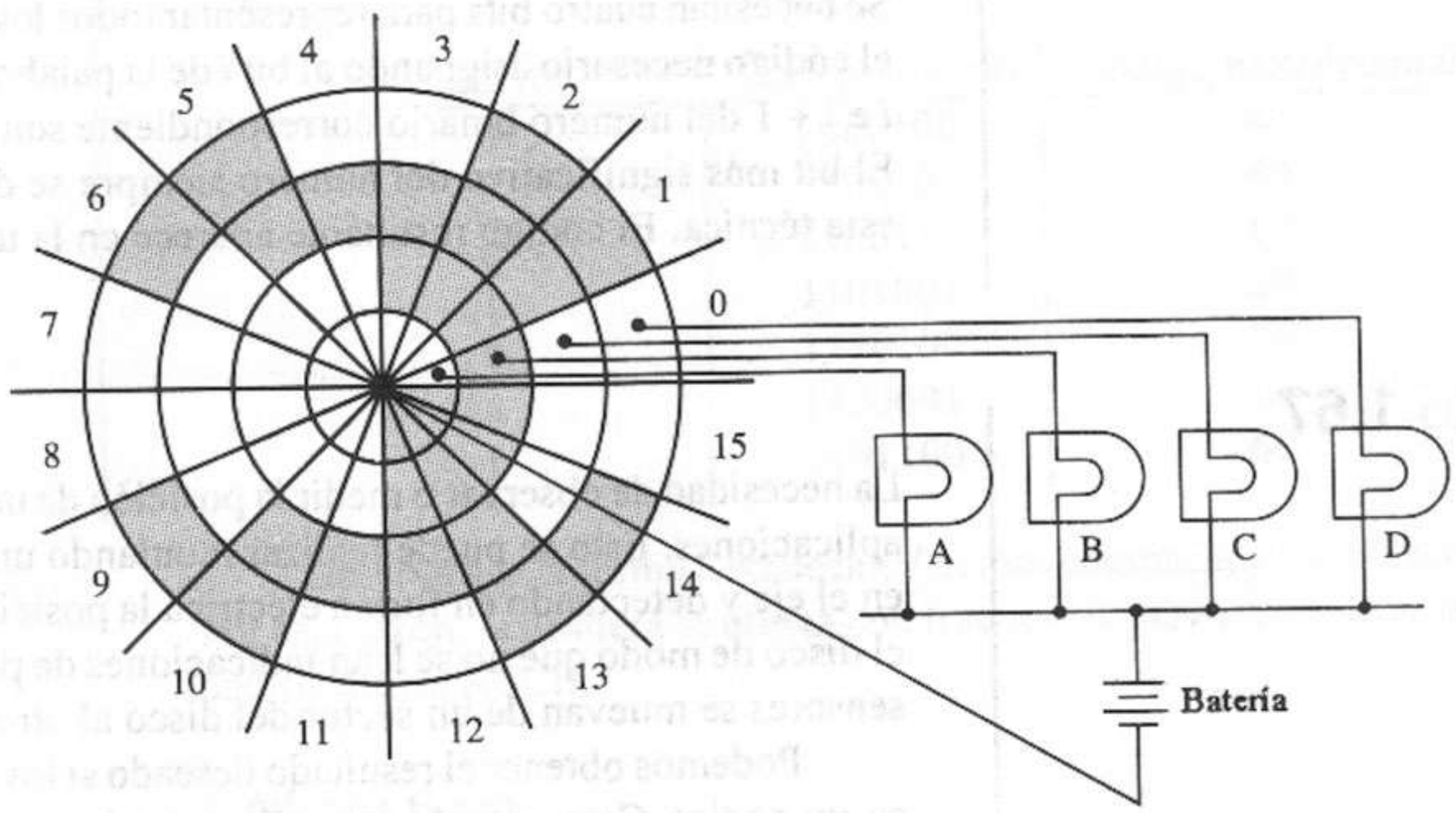
EJEMPLO 1.67

La necesidad de observar o medir la posición de un eje circular ocurre en muchas aplicaciones. Esto se puede realizar montando un círculo conductor codificado en el eje y detectando en forma eléctrica la posición del disco. ¿Cómo codificar el disco de modo que no se lean indicaciones de posición incorrectas cuando los sensores se muevan de un sector del disco al otro?

Podemos obtener el resultado deseado si los sectores del disco se codifican *en un código Gray, puesto que sólo cambiará un bit de posición en el código si los sensores se mueven de un sector al siguiente. La figura 1.5 ilustra la* solución.

TABLA 1.12 CÓDIGO GRAY PARA LOS NÚMEROS DECIMALES DE 0 A 15

Decimal	Binario	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000



1.5.3 Códigos para detección y corrección de errores

Un *error* en un dato binario se define como un valor incorrecto en uno o más bits. Un *error simple* es un valor incorrecto en un solo bit, mientras que un *error múltiple* se refiere a la existencia de uno o más bits incorrectos. Los errores pueden deberse a fallas del hardware, interferencia externa (ruido) u otros eventos no deseados. La información se puede codificar mediante códigos especiales que permitan la detección e incluso la corrección de ciertas clases de errores. A continuación ilustramos algunos códigos sencillos de detección y corrección de errores.

Será útil establecer algunas definiciones y notaciones antes de presentar códigos específicos. Sean I y J palabras de información binaria de n bits. El *peso* de I, $w(I)$, se define como el número de bits de I iguales a 1. La *distancia* entre I y J, $d(I, J)$, es igual al número de posiciones de bit en que difieren I y J.

EJEMPLO 1.68

Determinar los pesos de I y J y la distancia entre ellos si I = (01101100) y J = (11000100).

Si contamos los bits 1 en cada número, tenemos que

$$w(I) = 4 \text{ y } w(J) = 3$$

A continuación, comparamos los dos números bit por bit, observando que difieren de la manera siguiente:

$$\begin{array}{cccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \uparrow & & \uparrow & & \uparrow & & & \end{array}$$

Los números difieren en posiciones de tres bits. Por tanto,

$$d(I, J) = 3$$

Propiedades generales de los códigos de detección y corrección de errores

Si la distancia entre dos palabras de código de un código C es mayor o igual que d_{min}, se dice que el código tiene *distancia mínima* d_{min}. Las propiedades de detección y corrección de errores de un código quedan determinadas en parte por su distancia mínima. Esto se ilustra en la figura 1.6, donde los puntos encerrados en un círculo representan palabras de código válidas y los no encerrados representan palabras que tienen errores. Unimos dos puntos si las palabras correspondientes difieren en exactamente una posición. Para un d_{min} dado, al menos se necesitan d_{min} errores para transformar una palabra de código válida en otra. Si hay menos de d_{min} errores, entonces se obtiene una palabra que no es del código y se puede detectar. Si la palabra que no es del código está más "cerca" de un código válido que de otro, se puede deducir la palabra de código original, y así corregirse el error.

En general, un código permite *corregir t errores* y detectar s errores adicionales si y sólo si se cumple la siguiente desigualdad.

$$2t + s + 1 \leq d_{min} \tag{1.25}$$

Un análisis de la ecuación 1.25 muestra que un código con detección de errores simples ($s = 1, t = 0$) requiere una distancia mínima de 2; un código con corrección de errores simples ($s = 0, t = 1$) requiere una distancia mínima de 3, y un código con corrección de errores simples y detección de errores dobles ($s = t = 1$) requiere una distancia mínima de 4. La figura 1.6 ilustra estas y otras combinaciones.

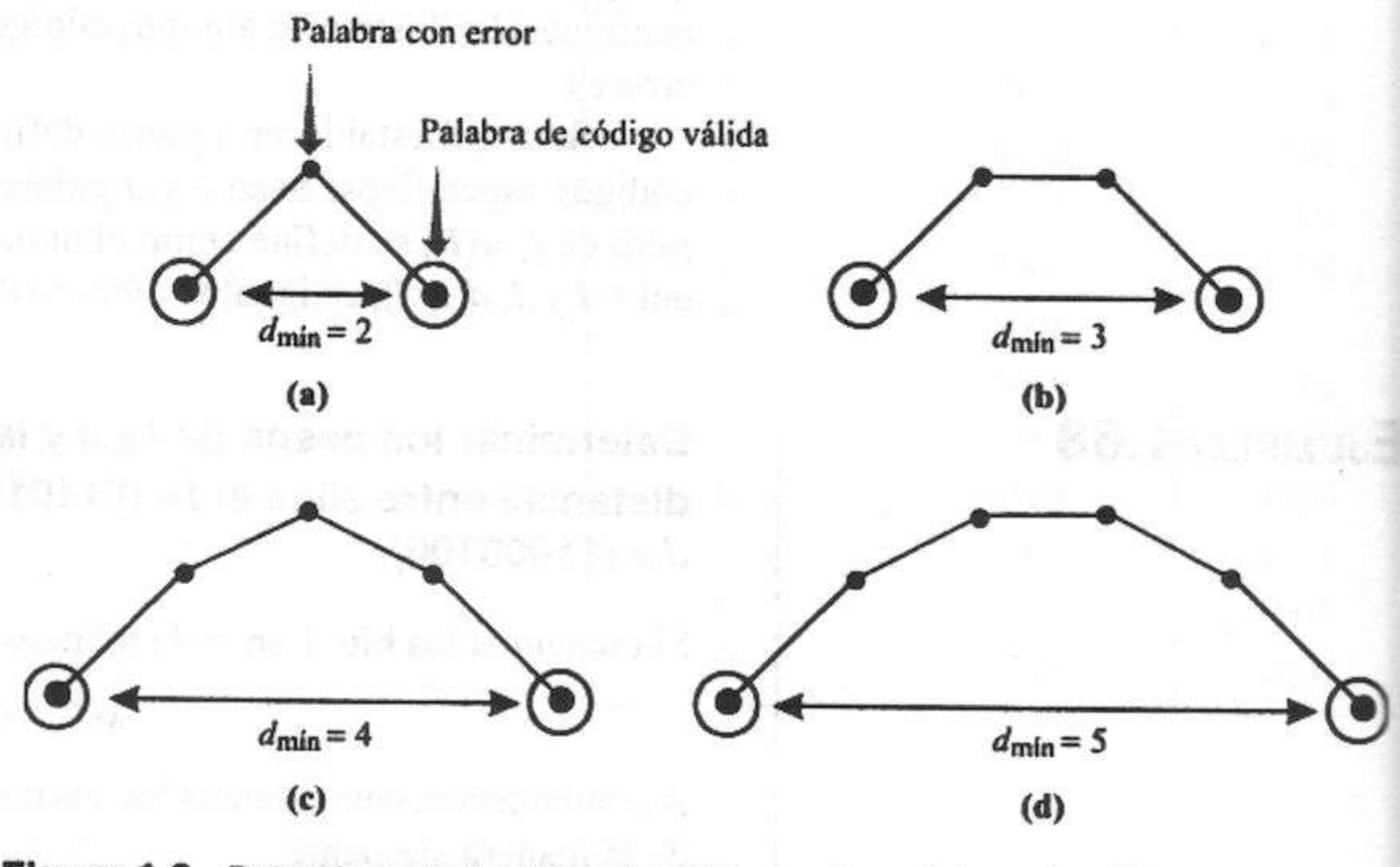


Figura 1.6 Relación entre la distancia mínima entre palabras de código y la capacidad para detectar y corregir errores de bit. (Las palabras unidas difieren exactamente en una posición de bit.) **(a)** Detección de errores simples (SED). **(b)** Corrección de errores simples (SEC) o corrección de errores dobles (DED). **(c)** (SEC y DED) o TED. **(d)** DEC (SEC y 3ED), o 4ED.

Códigos de paridad sencillos

Los códigos de paridad se forman a partir de un código C, concatenando (|) un *bit de paridad, P,* con cada palabra de código de C. La figura 1.7 ilustra el concepto. En un *código de paridad impar*, el bit de paridad se especifica como 0 o 1 de modo que $w(P \mid C)$ sea impar. El bit de paridad de un *código de paridad par* se selecciona de modo que $w(P \mid C)$ sea par. La figura 1.8 muestra la forma en que se utiliza la codificación con paridad en una cinta magnética de nueve pistas.

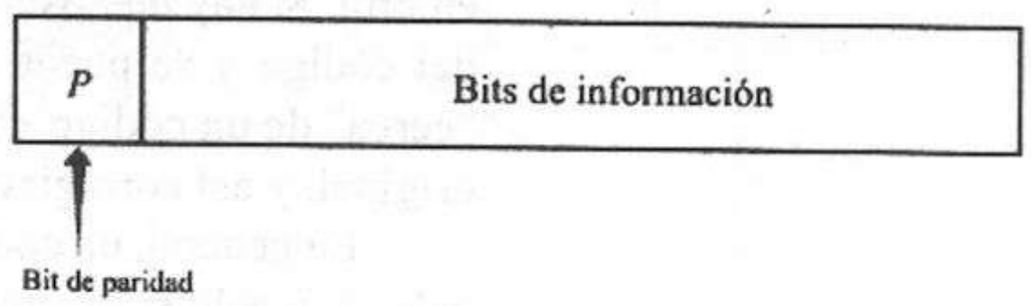


Figura 1.7 Información codificada con paridad.

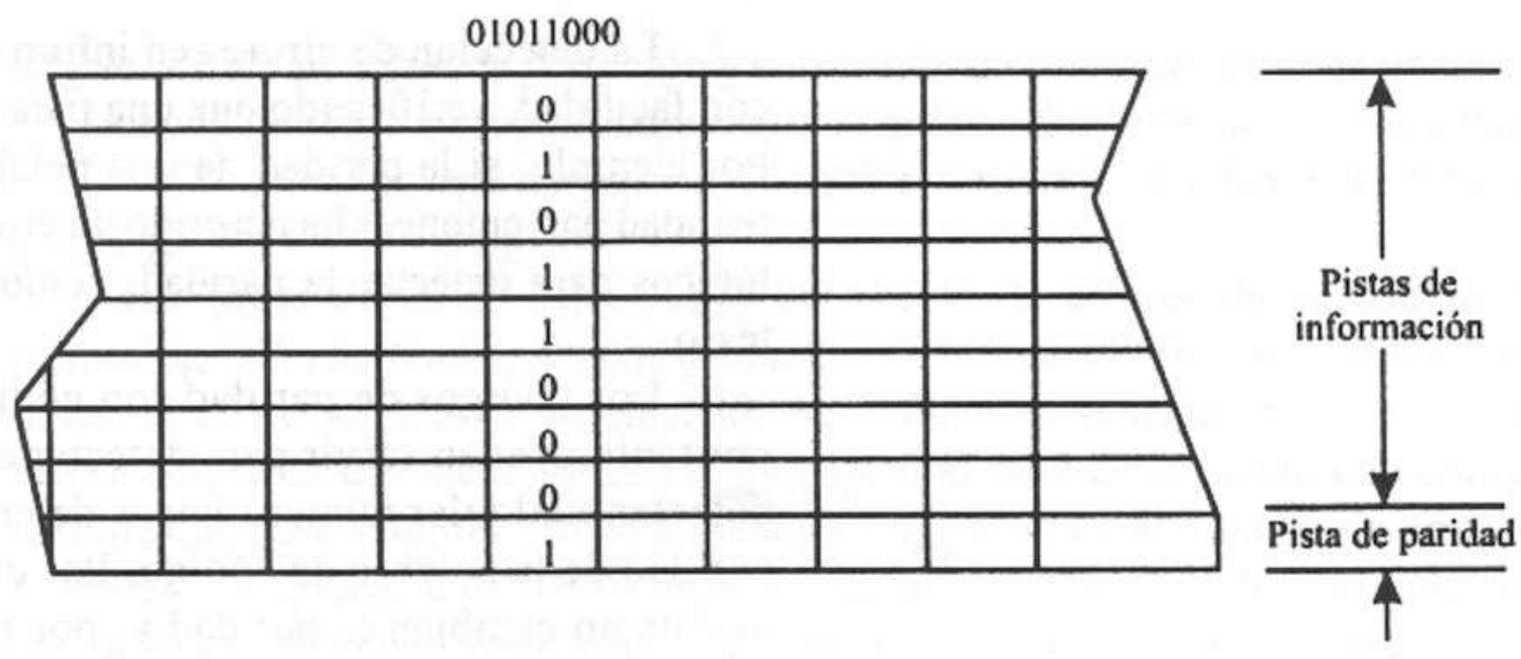


Figura 1.8 Codificación con paridad en cinta mágnetica.

EJEMPLO 1.69

Concatenar un bit de paridad con el código ASCII de los caracteres 0, *X*, = y BEL para obtener un código con paridad impar.

Carácter	Código ASCII	Código con paridad impar
0	0110000	10110000
X	1011000	01011000
=	0111100	10111100
BEL	0000111	00000111

EJEMPLO 1.70

Codificar el mensaje *CATCH 22* en código ASCII con paridad par y agrupar la palabra codificada en segmentos de 16 bits.

Segmento 1: $(\underbrace{11000011}_{C}\,\underbrace{01000001}_{A})_{\text{ASCII}}$

Segmento 2: $(\underbrace{11010100}_{T}\,\underbrace{11000011}_{C})_{\text{ASCII}}$

Segmento 3: $(\underbrace{01001000}_{H}\,\underbrace{10100000}_{Blanco})_{\text{ASCII}}$

Segmento 4: $(\underbrace{10110010}_{2}\,\underbrace{10110010}_{2})_{\text{ASCII}}$

Observe que este mensaje se puede guardar en cuatro palabras de memoria de una computadora de 16 bits como

Palabra X	1100001101000001
Palabra $X + 1$:	1101010011000011
Palabra $X + 2$:	0100100010100000
Palabra $X + 3$:	1011001010110010

La detección de errores en información codificada con paridad se realiza con facilidad, verificando que una palabra de código tenga la paridad correcta. Por ejemplo, si la paridad de una palabra de código con paridad impar es en realidad par, entonces ha ocurrido un error detectable. Es fácil construir circuitos lógicos para detectar la paridad, como veremos en secciones posteriores del texto.

Los códigos de paridad son códigos con distancia mínima igual a 2 y por tanto pueden servir para detectar errores simples. De hecho, sirven para detectar cualquier número impar de errores, pues tales errores cambiarán la paridad de la palabra de código. Por otro lado, los errores en un número par de bits no cambian la paridad y, por tanto, no se pueden detectar mediante un código de paridad.

Código dos de cinco

El *código dos de cinco* es un código para detección de errores que tiene exactamente 2 bits iguales a 1 y 3 bits iguales a 0 en cada palabra de código, y es representativo de los códigos *m* de *n*. La detección de errores se realiza contando el número de unos de una palabra de código. Se nota un error cuando el número de unos no es exactamente igual a 2. Esto implica que los códigos dos de cinco permiten la detección de errores simples y también de errores múltiples en bits adyacentes. La tabla 1.13 presenta un código dos de cinco para los dígitos decimales.

Códigos de Hamming

En 1950, Richard Hamming publicó la descripción de una clase de códigos para corrección de errores que han tenido amplio uso. Los *códigos de Hamming* se pueden ver como una extensión de los códigos de paridad simple, en el sentido de que se utilizan varios *bits* de paridad o *bits de verificación*. Cada bit de verificación se define sobre un subconjunto de los bits de información de una palabra. Los subconjuntos se traslapan de modo que cada bit de información está en al menos dos subconjuntos. Los códigos para *corrección de errores*

TABLA 1.13 CÓDIGOS DOS DE CINCO PARA LOS DÍGITOS DECIMALES

Dígito	Código dos de cinco
0	00011
1	00101
2	01001
3	10001
4	00110
5	01010
6	10010
7	01100
8	10100
9	11000

simples (SEC) permiten la detección y corrección de cualquier error en un único bit. Los códigos para *corrección de errores simples y detección de errores dobles (SEC/DED)* permiten detectar pero no corregir errores dobles, además de detectar y corregir los errores simples.

Las propiedades de detección y corrección de errores de un código de Hamming están determinadas por el número de bits de verificación utilizados y la forma como los bits de verificación se definen en relación con los bits de información. La distancia mínima d_{min} es igual al peso de la palabra de código no nula con peso mínimo. En otras palabras, d_{min} es igual al número de unos de la palabra de código con menos unos. No es un objetivo de este libro analizar con detalle el diseño de los códigos de Hamming. Sin embargo, utilizaremos los dos códigos de Hamming que aparecen en la tabla 1.14 para ilustrar las propiedades de estos códigos. Además, presentaremos un método para el diseño de códigos de Hamming SEC sencillos.

Código de Hamming 1. El código permite corregir errores simples pero no detectar errores dobles, pues su distancia mínima es 3. Esto se puede ver con claridad en el siguiente análisis. Un error simple en el bit extremo izquierdo de la palabra de código 0100110 produce la palabra errónea 1100110. La tabla 1.15 muestra la diferencia y la distancia entre cada palabra de código válida y la palabra errónea.

Observe que sólo la palabra de código donde ha ocurrido el error tiene una distancia 1 con respecto a la palabra errónea. Esto significa que ningún error simple en cualquier otra palabra de código podría haber producido la palabra

TABLA 1.14 DOS CÓDIGOS DE HAMMING PARA PALABRAS DE INFORMACIÓN DE CUATRO BITS

Palabras de información $(i_3i_2i_1i_0)$	Código de Hamming 1 $(i_3i_2i_1i_0c_2c_1c_0)$	Código de Hamming 2 $(i_3i_2i_1i_0c_3c_2c_1c_0)$
0000	0000000 0	0 00000000 0
0001	0001011 3	1 00011011 11
0010	0010101 5	2 00101101 13
0011	0011110 6	3 00110110 6
0100	0100110 6	4 01001110 14
0101	0101101 5	5 01010101 5
0110	0110011 3	6 01100011 3
0111	0111000 0	7 01111000 8
1000	1000111 7	8 10000111 7
1001	1001100 4	9 10011100 12
1010	1010010 2	10 10101010 10
1011	1011001 1	11 10110001 1
1100	1100001 1	12 11001001 9
1101	1101010 2	13 11010010 2
1110	1110100 4	14 11100100 4
5 1111	1111111 7	15 11111111 15

errónea. Por tanto, la detección de la palabra errónea 1100110 equivale a corregir el error, pues el único error simple posible que puede producir el patrón es un error en el bit extremo izquierdo de la palabra de código 0100110.

El análisis anterior también sugiere un procedimiento de detección y corrección de errores. Es decir, podríamos determinar la diferencia entre una palabra de datos y cada palabra de código válida posible. Una distancia de 0 indicaría una concordancia válida, una distancia de 1 indicaría un error simple en la palabra de código correspondiente, en la posición que corresponde al bit 1 de la diferencia, y una distancia mayor o igual que 2 respecto a todas las palabras de código indicaría un error múltiple. Aunque este procedimiento funciona en teoría, no sería práctico para códigos con un gran número de palabras de código. Más adelante analizaremos algunos métodos prácticos.

Nuestro análisis también revela que varias palabras de código tienen una distancia 2 con respecto a la palabra errónea. Por tanto, un error doble en cualquiera de estas palabras produciría la misma palabra errónea que el error simple (examine la figura 1.6). Esto implica que, en general, los errores dobles no se pueden detectar con este código. La corrección de errores simples aunada a la detección de errores dobles requiere un código con una distancia mínima de 4.

Los bits de verificación del código se definen de modo que den una paridad par con un subconjunto de los bits de información, como sigue:

c_2: i_3, i_2, i_1
c_1: i_3, i_2, i_0
c_0: i_3, i_1, i_0

TABLA 1.15 EFECTOS DE LOS ERRORES SOBRE LAS PALABRAS DE CÓDIGO

Palabras de código	Palabra errónea	Diferencia	Distancia
0000000	1100110	1100110	4
0001011	1100110	1101101	5
0010101	1100110	1110011	5
0011110	1100110	1111000	4
0100110	1100110	1000000	1
0101101	1100110	1001011	4
0110011	1100110	1010101	4
0111000	1100110	1011110	5
1000111	1100110	0100001	2
1001100	1100110	0101010	3
1010010	1100110	0110100	3
1011001	1100110	0111111	6
1100001	1100110	0000111	3
1101010	1100110	0001100	2
1110100	1100110	0010010	2
1111111	1100110	0011001	3

Esta relación se puede especificar de manera conveniente mediante una matriz conocida como *matriz generatriz*, o matriz G, como se muestra a continuación. Cada columna de la matriz G corresponde a un bit de la palabra de código, según se indica.

$$G = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \\ \begin{array}{ccccccc} i_3 & i_2 & i_1 & i_0 & c_2 & c_1 & c_0 \end{array} \end{array} = \begin{bmatrix} 1 & 0 & 0 & 0 & p_{11} & p_{12} & p_{13} \\ 0 & 1 & 0 & 0 & p_{21} & p_{22} & p_{23} \\ 0 & 0 & 1 & 0 & p_{31} & p_{32} & p_{33} \\ 0 & 0 & 0 & 1 & p_{41} & p_{42} & p_{43} \end{bmatrix} \quad (1.26)$$

La codificación de una palabra de información i para obtener una palabra de código c se puede expresar en términos de la matriz generatriz G como sigue.

$$c = iG \quad (1.27)$$

La decodificación de una palabra de código se puede expresar de manera conveniente en términos de una matriz H conocida como *matriz de verificación de paridad*. La matriz H se puede deducir de la matriz G como sigue, para el código anterior.

$$H = \begin{bmatrix} p_{11} & p_{21} & p_{31} & p_{41} & 1 & 0 & 0 \\ p_{12} & p_{22} & p_{32} & p_{42} & 0 & 1 & 0 \\ p_{13} & p_{23} & p_{33} & p_{43} & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (1.28)$$

Una n–tupla c es una palabra de código generada por G si y sólo si

$$Hc^T = 0 \quad (1.29)$$

Sea d una palabra de datos correspondiente a una palabra de código c, que ha sido corrompida por un patrón de error e. Entonces

$$d = c + e \quad (1.30)$$

La decodificación comienza con el cálculo del *síndrome* s de d para determinar si existe un error. Si no hay error, la decodificación concluye al eliminarse los bits de verificación, dejando sólo los bits de información originales. Si se encuentra un error corregible, se corrige antes de eliminar los bits de verificación. Si se encuentra un error no corregible, el proceso termina con una señal de error que indica la situación.

El síndrome de d se calcula como sigue, utilizando H:

$$\begin{aligned} s &= Hd^T \quad (1.31) \\ &= H(c+e)^T \\ &= Hc^T + He^T \\ &= 0 + He^T \\ &= He^T \quad (1.32) \end{aligned}$$

Los síndromes para la matriz H dada en la ecuación 1.28 aparecen en la tabla 1.16. Observe que el patrón de cada síndrome es igual al patrón de la columna de la matriz H que corresponde al bit erróneo.

Código de Hamming 2. La distancia mínima es 4, ya que ninguna palabra de código no nula tiene un peso menor que 4. Por tanto, el código tiene las

TABLA 1.16 SÍNDROMES Y PATRONES DE ERROR

Patrón de error	Síndrome	Significado
0 0 0 0 0 0 0	0 0 0	Sin error
0 0 0 0 0 0 1	0 0 1	Error en c_0
0 0 0 0 0 1 0	0 1 0	Error en c_1
0 0 0 0 1 0 0	1 0 0	Error en c_2
0 0 0 1 0 0 0	0 1 1	Error en i_0
0 0 1 0 0 0 0	1 0 1	Error en i_1
0 1 0 0 0 0 0	1 1 0	Error en i_2
1 0 0 0 0 0 0	1 1 1	Error en i_3

propiedades de corrección de errores simples y detección de errores dobles. Las matrices generatriz y de verificación de paridad son las siguientes:

$$G = \begin{bmatrix} 10000111 \\ 01001110 \\ 00101101 \\ 00011011 \end{bmatrix} \tag{1.33}$$

$$H = \begin{bmatrix} 01111000 \\ 11100100 \\ 11010010 \\ 10110001 \end{bmatrix} \tag{1.34}$$

Observe que cada columna de la matriz H de la ecuación 1.34 tiene un número impar de unos. Tales códigos de Hamming se llaman códigos con *columnas de peso impar* y tienen varias propiedades deseables, incluidas la corrección de errores simples, la detección de errores dobles y la detección de otros errores múltiples. Además, permiten el uso de circuitos de codificación y decodificación relativamente económicos y rápidos. Es por esto que, los códigos con columnas de peso impar se utilizan con frecuencia en la práctica.

La forma más fácil de diseñar un código de Hamming es especificando la matriz H. Para cualquier entero positivo $m \geq 3$, existe un código SEC (m,k) con las siguientes propiedades:

- Longitud de código: $n = 2^m - 1$
- Número de bits de información: $k = 2^m - m - 1$
- Número de bits de verificación: $n - k = m$
- Distancia mínima: $d_{min} = 3$

La matriz H para tal código es una matriz $n \times m$ formada por todas las m–tuplas binarias distintas de cero como columnas. La matriz de la ecuación 1.28 es un ejemplo de tales matrices, con $m = 3$. Observe que podemos determinar las demás matrices H para $m = 3$ reordenando las columnas.

Obtenemos un código de Hamming (15,11) cuando $m = 4$. Una posible matriz H para tal código es la siguiente:

$$H = \begin{bmatrix} 111101110001000 \\ 111011001100100 \\ 110110100110010 \\ 101110011010001 \end{bmatrix} \tag{1.35}$$

Podemos eliminar l columnas arbitrarias de una matriz H de un código de Hamming para obtener otro código de Hamming con las siguientes propiedades:

- Longitud de código: $n = 2^m - l - 1$
- Número de bits de información: $k = 2^m - m - l - 1$
- Número de bits de verificación: $n - k = m$
- Distancia mínima: $d_{min} \geq 3$

Estas propiedades conducen a la posibilidad de diseñar códigos con mejores propiedades de detección y corrección de errores y longitudes de código más útiles.

EJEMPLO 1.71

Diseñar un código de Hamming para codificar cinco bits de información ($k = 5$).

Se necesitan cuatro bits de verificación ($m = 4$), ya que para $m = 3$, $k = 2^3 - 3 - 1 = 4 < 5$. Sin embargo, para $m = 4$, $k = 2^4 - 4 - 1 = 11 > 5$. Pero podemos obtener un código (9,5) eliminando seis columnas de la matriz H de un código (15,11). Al eliminar seis columnas de la ecuación 1.35 obtenemos

$$H = \begin{bmatrix} 111101000 \\ 111010100 \\ 110110010 \\ 101110001 \end{bmatrix} \tag{1.36}$$

La matriz generatriz correspondiente es

$$G = \begin{bmatrix} 100001111 \\ 010001110 \\ 001001101 \\ 000101011 \\ 000010111 \end{bmatrix} \tag{1.37}$$

Esto concluye nuestro estudio de los códigos de detección y corrección de errores. Los lectores que deseen aprender más acerca de estos códigos pueden consultar la referencia bibliográfica [4].

1.6 Resumen

Hemos completado nuestra introducción a los sistemas numéricos y los códigos de computadoras. El lector debe estar ahora familiarizado con los sistemas numéricos decimal, binario, octal y hexadecimal y podrá convertir los números de cualquiera de estas bases a cualquier otra. Además, el lector deberá ya entender las operaciones aritméticas en todas las bases y la forma en que pueden

representarse los números negativos en las computadoras. También, deberá haber adquirido cierta familiaridad con los números de punto fijo y de punto flotante, y comprenderá en general los códigos de caracteres decimal codificado en binario (BCD) y ASCII. Así mismo, presentamos los códigos Gray y de exceso o sesgados. Por último, el lector deberá haber adquirido un conocimiento general de los códigos de detección y corrección de errores simples. Puede lograr una comprensión más detallada de estos temas consultando la bibliografía.

REFERENCIAS BIBLIOGRÁFICAS

1. M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes, "*IBM Journal of Research and Development*, Vol. 14, No. 4, págs. 395-401, julio 1970.
2. K. Hwang, *Computer Arithmetic*. Nueva York: Wiley, 1979.
3. D.E.Knuth, *Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1969.
4. S. Lin y D.J. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1983.
5. W.W. Peterson y E.J. Weldon, Jr., *Error-correcting Codes*, 2a. edición Cambridge, MA: MIT Press, 1972.
6. J.F. Wakerly, *Microcomputer Architecture and Programming*. Nueva York: Wiley, 1981.
7. S. Waser y M.J. Flynn, *Introduction to Arithmetic for Digital Systems*. Nueva York: Holt, Rinehart, and Winston, 1982.
8. *IEEE Standard for Binary Floating-point Arithmetic*, ANSI/IEEE Std. 754-1985. Institute of Electrical and Electronic Engineers, 345 East 47th St., Nueva York, NY, agosto 1985.
9. Israel, Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice Hall 1993.

PROBLEMAS

1.1 Calcule $A + B$, $A - B$, $A \times B$ y $A \div B$ para las siguientes parejas de números binarios.

(a) 10101, 1011
(b) 1011010, 101111
(c) 101, 1011
(d) 10110110, 01011011
(e) 1101011, 1010
(f) 1010101, 101010
(g) 10000, 1001
(h) 1011.0101, 110.11

1.2 Calcule $A + B$, $A - B$, $A \times B$ y $A \div B$ para las siguientes parejas de números octales.

(a) 372, 156
(b) 704, 230
(c) 1000, 777
(d) 423, 651

1.3 Calcule $A + B$, $A - B$, $A \times B$ y $A \div B$ para las siguientes parejas de números hexadecimales.

(a) 2CF3, 2B
(b) FFFF, 1000
(c) 9A5, D17
(d) 372, 156

1.4 Convierta los siguientes números decimales a números binarios, octales y hexadecimales.

(a) 27
(b) 915
(c) 0.375
(d) 0.65
(e) 174.25
(f) 250.8

1.5 Convierta los siguientes números binarios a números octales, hexadecimales y decimales utilizando el método de conversión más adecuado.

(a) 1101
(b) 101110
(c) 0.101
(d) 0.01101
(e) 10101.11
(f) 10110110.001

1.6 Convierta los siguientes números octales a números binarios, hexadecimales y decimales utilizando el método de conversión más adecuado.

(a) 65
(b) 371
(c) 240.51
(d) 2000
(e) 111111
(f) 177777

1.7 Convierta los siguientes números hexadecimales a números binarios, octales y decimales utilizando el método de conversión más adecuado.

(a) 4F
(b) ABC
(c) F8.A7
(d) 2000
(e) 201.4
(f) 3D65E

1.8 Determine el complemento a dos de los siguientes números binarios, suponiendo $n = 8$.

(a) 101010
(b) 1101011
(c) 0
(d) 11111111
(e) 10000000
(f) 11000

1.9 Determine el complemento a uno de los siguientes números binarios, suponiendo $n = 8$.

(a) 110101
(b) 1010011
(c) 0
(d) 10000000
(e) 100001
(f) 01111111

1.10 Calcule $A + B$, $A - B$, $-A + B$ y $-A - B$ para las siguientes parejas de números, suponiendo un sistema numérico de complemento a dos y $n = 8$. Verifique sus resultados mediante aritmética decimal. Explique los resultados extraños.

(a) 1010101, 1010
(b) 1101011, 0101010
(c) 11101010, 101111
(d) 10000000, 01111111

1.11 Repita el problema 1.10 para los siguientes números, utilizando un sistema numérico de complemento a uno.

(a) 1010101, 1101
(b) 10111010, 11010
(c) 1010101, 0101010
(d) 10000000, 01111111

1.12 Muestre la forma en que una computadora de 16 bits con un sistema numérico de complemento a dos realizaría los siguientes cálculos.

(a) $(16850)_{10} + (2925)_{10} = (?)_{10}$
(b) $(16850)_{10} - (2925)_{10} = (?)_{10}$
(c) $(2925)_{10} - (16850)_{10} = (?)_{10}$
(d) $-(2925)_{10} - (16850)_{10} = (?)_{10}$

1.13 Codifique los siguientes números en los códigos BCD y exceso 3.

(a) 39
(b) 1950
(c) 94704
(d) 625

1.14 Codifique las siguientes cadenas en código ASCII. Represente las cadenas codificadas mediante números hexadecimales.

(a) 1980
(b) A = b + C
(c) COMPUTER ENGINEERING
(d) The End.

1.15 Defina un código de cuatro bits para la representación de dígitos decimales, con la propiedad de que las palabras de código para cualesquiera dos dígitos cuya diferencia sea uno difieran sólo en una posición de bit, y que esto también se cumpla para los dígitos 0 y 9.

1.16 ¿Cuántos errores de bit se pueden detectar en un código dos de cinco? ¿Cuántos errores, si acaso, se pueden corregir en un código dos de cinco? Demuestre sus respuestas en forma matemática.

1.17 Examine el disco con código Gray de la figura 1.5. Suponga que los indicadores señalan lo siguiente: *A* está desactivado, *B* está activado, *C* está activado y *D* está parpadeando. Localice la posición del disco según los números de sector.

1.18 Se grabarán los siguientes mensajes de ocho bits en la cinta magnética de nueve pistas de la figura 1.7. Determine el bit de paridad para que cada mensaje tenga paridad impar.

(a) P10111010
(b) P00111000
(c) P10011001
(d) P01011010

1.19 Sea 10111001 una palabra errónea obtenida mediante el código de Hamming 2. Determine la palabra de código correcta, calculando la diferencia y la distancia entre la palabra errónea y cada palabra de código válida.

1.20 Desarrolle una tabla de síndromes para el código de Hamming 2 que considere el caso libre de errores, exclusivamente errores simples y exclusivamente errores dobles. ¿Existe una caracterización sencilla de los síndromes de errores dobles? ¿Existen patrones de error con tres o más errores detectables por el código?

1.21 Utilice la tabla de síndromes desarrollada en el problema 1.20 para decodificar las siguientes palabras.

(a) 10010111
(b) 10011011
(c) 00111110
(d) 00000111
(e) 11101110
(f) 01011000
(g) 11100001
(h) 01101000

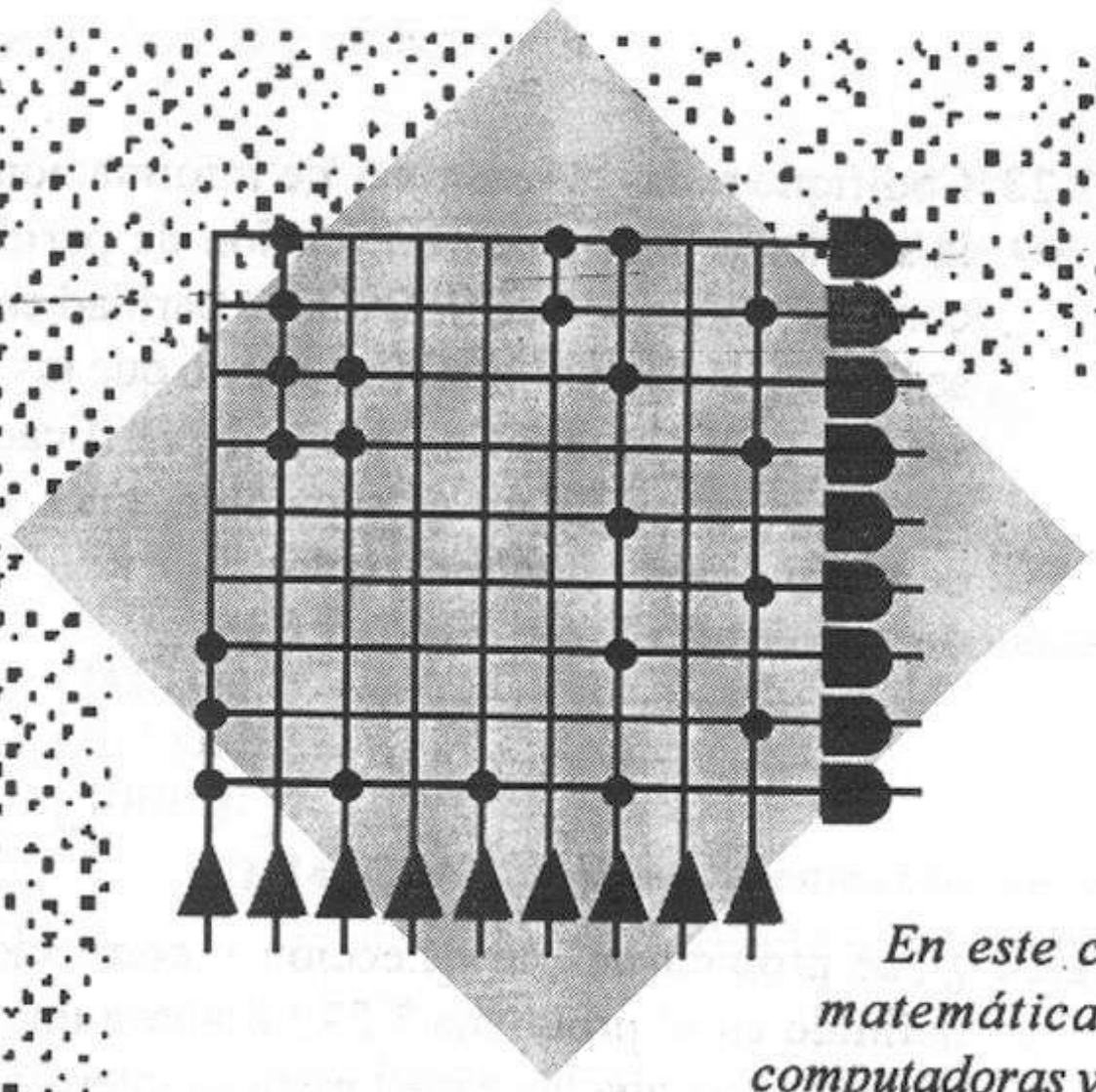
1.22 Desarrolle las matrices generatriz y de verificación de paridad para el código SEC de Hamming, codificando palabras de información con una longitud de 6 bits.

1.23 Codifique todas las palabras de información para un código definido por la siguiente matriz de verificación de paridad. Observe que los códigos con una matriz de verificación de paridad en la forma de la ecuación 1.28 se llaman códigos separables, puesto que los bits de información se pueden separar como bloque de los bits de verificación. El código que resulta de la siguiente matriz no es separable, pues los bits de información y los de verificación están mezclados.

$$H = \begin{bmatrix} 1111000 \\ 1100110 \\ 1010101 \end{bmatrix}$$

1.24 ¿Qué propiedades de detección y corrección de errores tiene el código definido en el problema 1.23? Elabore una tabla de síndromes para este código. Describa las características interesantes de los síndromes.

1.25 Describa las ventajas y desventajas de los códigos separables que tienen la forma representada por la matriz de la ecuación 1.28, en comparación con los códigos no separables de la forma representada por la matriz del problema 1.23.



2

En este capítulo presentamos las herramientas matemáticas básicas para el diseño lógico de computadoras y los conceptos matemáticos subyacentes. El material no sólo es en sí un tema importante, sino que también proporciona la base para los conceptos avanzados que se analizarán en secciones posteriores del texto. El material de este capítulo pretende ser independiente de los elementos de circuito específicos utilizados para construir circuitos digitales. En los siguientes capítulos analizaremos la aplicación de estas herramientas matemáticas a diversos tipos de elementos de circuitos.

Métodos algebraicos para el análisis y síntesis de circuitos lógicos

2.1 Fundamentos del álgebra booleana

Las herramientas de análisis y síntesis presentadas en este capítulo se basan en los conceptos fundamentales del álgebra booleana y, por tanto, analizaremos ahora este tema. En 1849, George Boole presentó una formulación algebraica de los procesos del pensamiento y el razonamiento lógico [1]. Esta formulación se conoce como álgebra booleana, que resumiremos a continuación.

2.1.1 Postulados básicos

La descripción básica de la formulación del álgebra booleana se basa en conceptos de la teoría de conjuntos, donde se define formalmente un *álgebra booleana* como un conjunto matemático distributivo y complementado [2]. Resumiremos aquí esta definición mediante un conjunto de postulados que sintetiza los elementos y propiedades básicos de un álgebra booleana.

Postulado 1. Definición Un *álgebra booleana* es un sistema algebraico cerrado formado por un conjunto K de dos o más elementos y los dos operadores $\bullet$ y $+$; de manera alternativa, para cada a y b un conjunto K, $a \bullet b$ pertenece a K y $a + b$ pertenece a K ($+$ se llama OR y $\bullet$ se llama AND).

Postulado 2. Existencia de los elementos 1 y 0 En el conjunto K existen los elementos 1 (uno) y 0 (cero), únicos, tales que para toda a en K

(a) $a + 0 = a$,

(b) $a \bullet 1 = a$,

donde 0 es el elemento neutro para la operación $+$ y 1 es el elemento neutro para la operación $\bullet$.

Postulado 3. Conmutatividad de las operaciones + y • Para toda a y b en K

(a) $a + b = b + a$,

(b) $a \bullet b = b \bullet a$.

Postulado 4. Asociatividad de las operaciones + y • Para toda a, b y c en K

(a) $a + (b + c) = (a + b) + c$,

(b) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

Postulado 5. Distributividad de + sobre • y de • sobre + Para toda a, b y c en K

(a) $a + (b \cdot c) = (a + b) \cdot (a + c)$,

(b) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

Postulado 6. Existencia del complemento Para toda a en K existe un único elemento llamado $\overline{a}$ (*complemento* de a) en K tal que

(a) $a + \overline{a} = 1$,

(b) $a \cdot \overline{a} = 0$.

Con este conjunto de premisas, podemos desarrollar otras relaciones útiles, que llamaremos teoremas. Para simplificar la notación en el resto del texto, suprimiremos el punto (•) al indicar la operación •.

EJEMPLO 2.1

$$a + b \cdot c = (a + b) \cdot (a + c)$$

$$a + bc = (a + b)(a + c)$$

Antes de desarrollar los teoremas, examinemos los postulados con detalle para entender exactamente lo que significan.

2.1.2 Diagramas de Venn para los postulados [2]

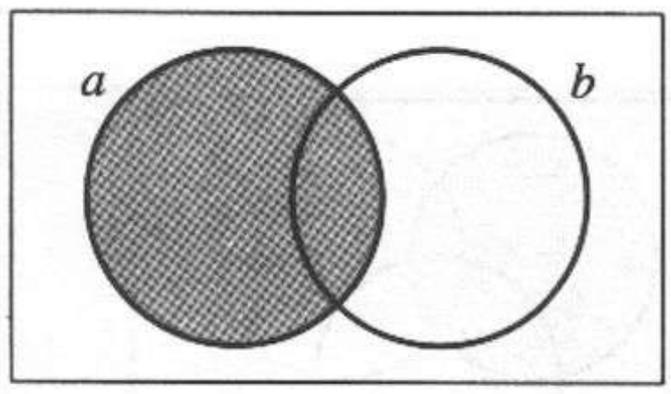
Podemos representar los postulados de manera gráfica en forma de diagramas de Venn. Esta descripción gráfica es posible, ya que el álgebra de conjuntos es un álgebra booleana en la que los conjuntos son los elementos del álgebra, la operación de intersección corresponde a • y la operación de unión corresponde a +. En el diagrama de Venn, los conjuntos se muestran como contornos cerrados, es decir, círculos, cuadrados, elipses, etc. Los diagramas de Venn para los conjuntos a, b, $a \cdot b$ y $a + b$ aparecen en la figura 2.1. Otras notaciones frecuentes para $a + b$ es $a \vee b$ o $a \cup b$ y para $a \cdot b$, ab, $a \wedge b$ o $a \cap b$.

Podemos utilizar los diagramas de Venn para ilustrar los postulados. Elegimos como ejemplo el postulado 5.

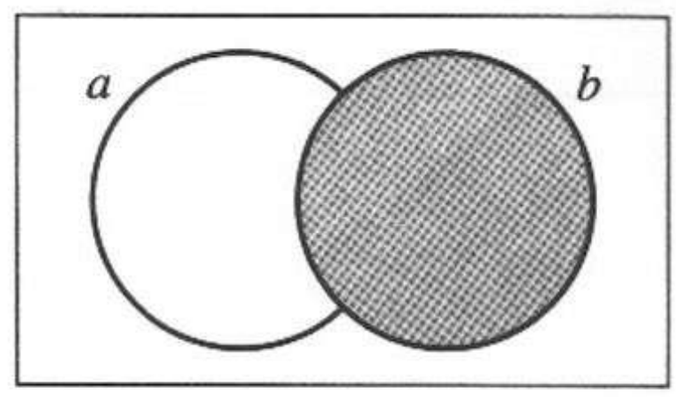
EJEMPLO 2.2

Utilizar el diagrama de Venn para ilustrar el postulado 5.

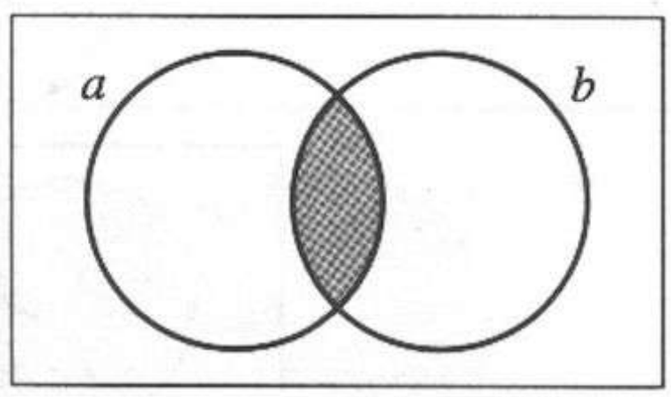
Del análisis de la figura 2.2, es evidente que el conjunto $a + bc$ y el conjunto $(a + b)(a + c)$ son dos representaciones de la misma área sombreada y, por tanto, $a + bc$ es igual a $(a + b)(a + c)$.



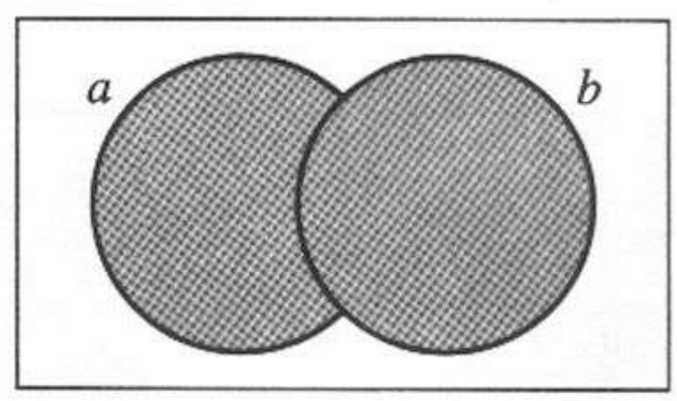
El conjunto a está sombreado.



El conjunto b está sombreado.



El conjunto $a \bullet b$ está sombreado.



El conjunto $a + b$ está sombreado.

Figura 2.1 Ejemplos de diagramas de Venn.

Es interesante examinar algunas facetas del postulado 6. Este postulado se refiere al complemento de a. Si a es el conjunto sombreado de la figura 2.3, el complemento de a, $\overline{a}$, es el área fuera de a en el conjunto universal. En otras palabras, a y $\overline{a}$ son mutuamente excluyentes y se encuentran dentro del conjunto universal. Como son mutuamente excluyentes, no tienen un área común y, por tanto, su intersección es el conjunto vacío: $a \bullet a = 0$. La unión de a y $\overline{a}$ es por definición el conjunto universal: $a + a = 1$.

Además, como el conjunto universal 1 contiene a todos los demás conjuntos, su complemento debe ser el conjunto vacío, 0. Por tanto, $\overline{1} = 0$ y $\overline{0} = 1$.

El diagrama de Venn es una herramienta muy útil no sólo para visualizar los postulados ya presentados, sino también los teoremas importantes del álgebra booleana que describimos a continuación.

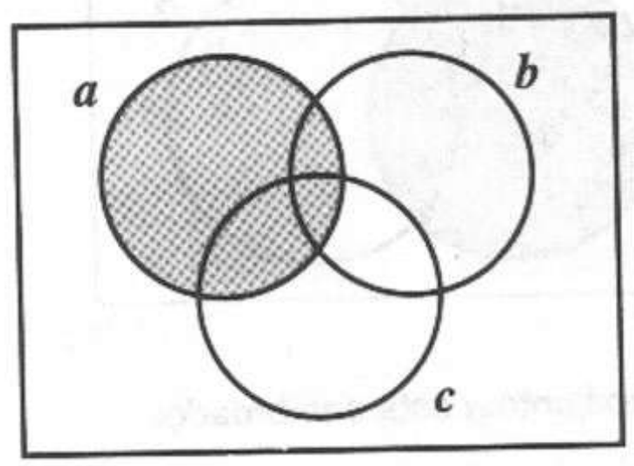
2.1.3 Dualidad

El principio de *dualidad* es un concepto muy importante en el álgebra booleana. En pocas palabras, el principio de dualidad establece que, si una expresión es válida en el álgebra booleana, entonces su expresión dual también es válida. Determinamos la expresión dual remplazando todos los operadores + por •, todos los operadores • por +, todos los unos por ceros y todos los ceros por unos.

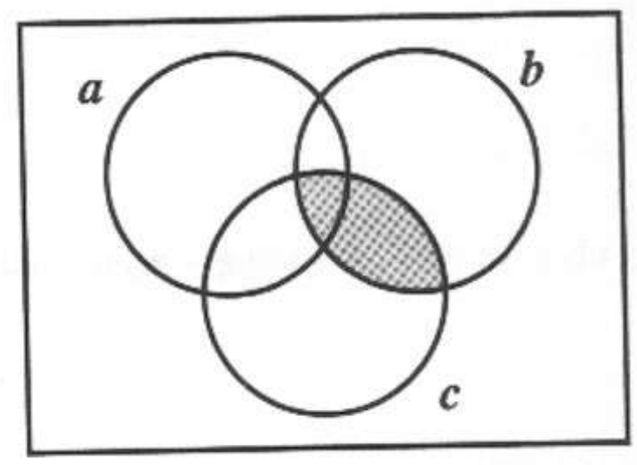
EJEMPLO 2.3

Determinar la expresión dual de

$$a + (bc) = (a + b)(a + c)$$

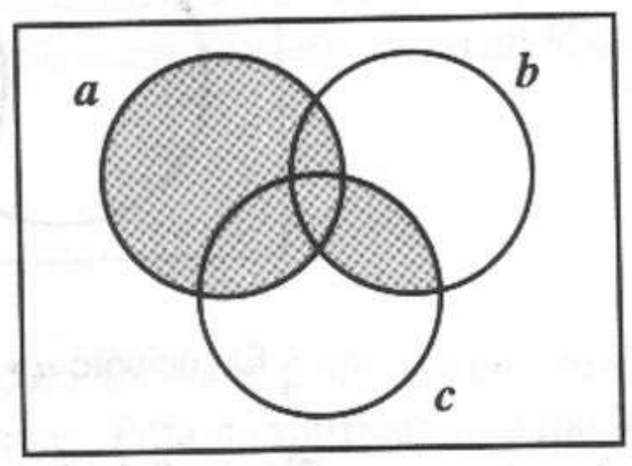


El conjunto a está sombreado.

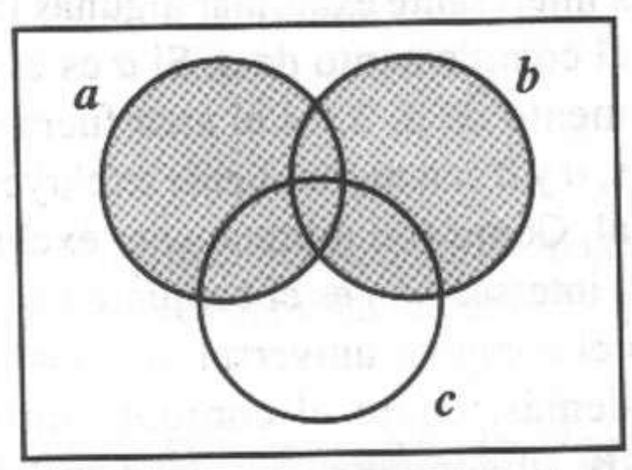


El conjunto $b \cdot c$ está sombreado.

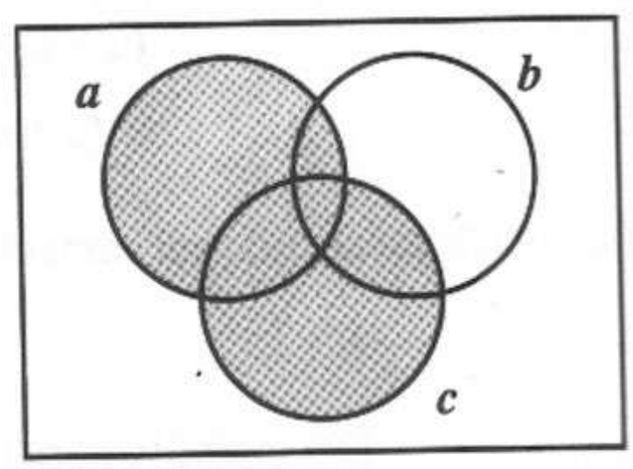
Unión



El conjunto $a + bc$ está sombreado.

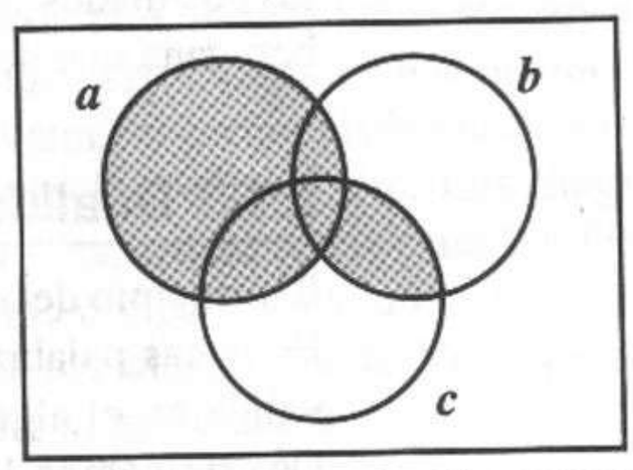


El conjunto $a + b$ está sombreado.



El conjunto $a + c$ está sombreado.

Intersección



El conjunto $(a + b)(a + c)$ está sombreado.

Figura 2.2 Diagramas de Venn para el postulado 5.

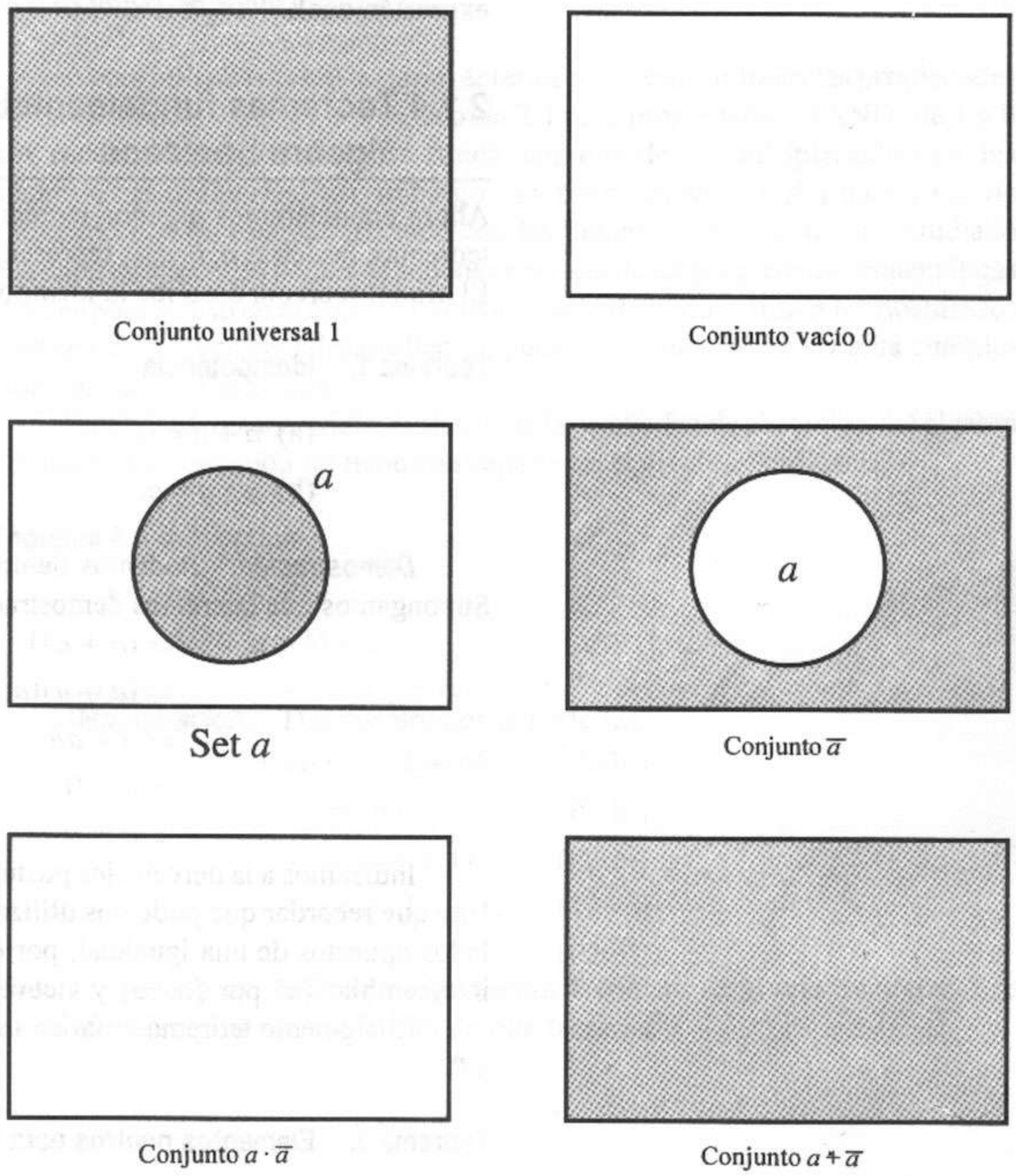


Figura 2.3 Diagramas de Venn que ilustran el postulado 6.

Al cambiar todos los operadores + por • y viceversa, obtenemos la expresión dual

$$a(b + c) = ab + ac$$

Al obtener el dual, no debemos alterar la posición de los paréntesis existentes. Observe que las dos expresiones del último ejemplo son las partes (a) y (b) del postulado 5. De hecho, presentamos los postulados 2 a 6 como expresiones duales.

Utilizaremos ampliamente el principio de dualidad al demostrar los teoremas del álgebra booleana. De hecho, una vez que hayamos utilizado los

postulados y los teoremas demostrados anteriormente para demostrar la validez de una expresión, podremos usar la dualidad para demostrar la validez de la expresión dual.

2.1.4 Teoremas fundamentales del álgebra booleana

Ahora enunciaremos algunos teoremas útiles del álgebra booleana. En estos teoremas, las letras a, b, c, ... representan los elementos de un álgebra booleana. El primer teorema describe la idempotencia y se expresa como sigue.

Teorema 1. Idempotencia

(a) $a + a = a$.

(b) $a \cdot a = a$.

Demostración. Podemos demostrar la parte (a) o (b) de este teorema. Supongamos que queremos demostrar la parte (a):

$$\begin{aligned} a + a &= (a + a)1 && [\text{P2(b)}] \\ &= (a + a)(a + \bar{a}) && [\text{P6(a)}] \\ &= a + a\bar{a} && [\text{P5(a)}] \\ &= a + 0 && [\text{P6(b)}] \\ &= a && [\text{P2(a)}] \end{aligned}$$

Indicamos a la derecha los postulados que justifican un paso en particular. Hay que recordar que podemos utilizar de manera indistinta las expresiones de lados opuestos de una igualdad; por ejemplo, el teorema 1 dice que podemos intercambiar $\{a\}$ por $\{a \cdot a\}$ y viceversa.

El siguiente teorema enfatiza las propiedades de los elementos únicos 1 y 0.

Teorema 2. Elementos neutros para los operadores + y •

(a) $a + 1 = 1$.

(b) $a \cdot 0 = 0$.

Demostración. De nuevo, demostraremos la parte (a) del teorema.

$$\begin{aligned} a + 1 &= (a + 1)1 && [\text{P2(b)}] \\ &= 1 \cdot (a + 1) && [\text{P3(b)}] \\ &= (a + \bar{a})(a + 1) && [\text{P6(a)}] \\ &= a + \bar{a} \cdot 1 && [\text{P5(a)}] \\ &= a + \bar{a} && [\text{P2(b)}] \\ &= 1 && [\text{P6(a)}] \end{aligned}$$

Como la parte (a) de este teorema es válida, el principio de dualidad implica que la parte (b) también es válida.

Teorema 3. Involución

$\overline{\bar{a}} = a$.

Demostración. Por el postulado 5, $a \cdot \overline{a} = 0$ y $a + \overline{a} = 1$. Por tanto, $\overline{a}$ es el complemento de a, y también a es el complemento de $\overline{a}$. Como el complemento de $\overline{a}$ es único, se sigue que $\overline{\overline{a}} = a$.

Ahora utilizaremos el material anterior para resumir todas las propiedades de los elementos únicos 1 y 0 en la tabla 2.1. Las propiedades • (AND) de 1 y 0 nos recuerdan las propiedades fundamentales de la multiplicación en las matemáticas ordinarias; sin embargo, las propiedades + (OR) hacen ver de inmediato que no estamos tratando con las matemáticas que hemos estudiado con anterioridad, y no podemos suponer ninguna de las propiedades matemáticas usuales para su uso en el álgebra booleana. Sólo debemos utilizar los postulados y teoremas que estamos desarrollando, pues trabajamos en un sistema completamente nuevo y diferente.

En el siguiente teorema establecemos la propiedad de absorción del álgebra booleana. La absorción no tiene contraparte en el álgebra "ordinaria".

Teorema 4. Absorción

(a) $a + ab = a$.

(b) $a(a + b) = a$.

Demostración. Demostraremos la parte (a).

$$\begin{aligned} a + ab &= a \cdot 1 + ab && [P2(b)] \\ &= a(1 + b) && [P5(b)] \\ &= a(b + 1) && [P3(b)] \\ &= a \cdot 1 && [T2(a)] \\ &= a && [P2(b)] \end{aligned}$$

Podemos visualizar fácilmente el teorema 4 con un diagrama de Venn. Los siguientes ejemplos ilustran el uso de este teorema.

EJEMPLO 2.4

$$(X + Y) + (X + Y)Z = X + Y \qquad [T4(a)]$$

EJEMPLO 2.5

$$A\overline{B}(A\overline{B} + \overline{B}C) = A\overline{B} \qquad [T4(b)]$$

EJEMPLO 2.6

$$A\overline{B}C + \overline{B} = \overline{B} \qquad [T4(a)]$$

TABLA 2.1 PROPIEDADES DE LOS ELEMENTOS 0 Y 1

OR	ADN	COMPLEMENTO
$a + 0 = a$	$a \cdot 0 = 0$	$\overline{0} = 1$
$a + 1 = 1$	$a \cdot 1 = a$	$\overline{1} = 0$

Los siguientes tres teoremas son similares a la absorción en el sentido de que pueden servir para eliminar elementos adicionales de una expresión booleana.

Teorema 5.

(a) $a + \bar{a}b = a + b$.
(b) $a(\bar{a} + b) = ab$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} a + \bar{a}b &= (a + \bar{a})(a + b) && \text{[P5(a)]} \\ &= 1 \cdot (a + b) && \text{[P6(a)]} \\ &= (a + b) \cdot 1 && \text{[P3(b)]} \\ &= (a + b) && \text{[P2(b)]} \end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 5 para simplificar las expresiones booleanas.

EJEMPLO 2.7

$$B + A\bar{B}\bar{C}D = B + A\bar{C}D \qquad \text{[T5(a)]}$$

EJEMPLO 2.8

$$\bar{Y}(X + Y + Z) = \bar{Y}(X + Z) \qquad \text{[T5(b)]}$$

EJEMPLO 2.9

$$(X + Y)(\overline{(X + Y)} + Z) = (X + Y)Z \qquad \text{[T5(b)]}$$

EJEMPLO 2.10

$$AB + \overline{(AB)}C\bar{D} = AB + C\bar{D} \qquad \text{[T5(a)]}$$

Teorema 6.

(a) $ab + a\bar{b} = a$.
(b) $(a + b)(a + \bar{b}) = a$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} ab + a\bar{b} &= a(b + \bar{b}) && \text{[P5(b)]} \\ &= a \cdot 1 && \text{[P6(a)]} \\ &= a && \text{[P2(b)]} \end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 6 para simplificar expresiones booleanas.

EJEMPLO 2.11

$$ABC + A\bar{B}C = AC \qquad \text{[T6(a)]}$$

EJEMPLO 2.12

$$(AD + B + C)(AD + \overline{(B + C)}) = AD \qquad \text{[T6(b)]}$$

EJEMPLO 2.13

Simplificar $(\bar{W}+\bar{X}+\bar{Y}+\bar{Z})(\bar{W}+\bar{X}+\bar{Y}+Z)$
$(\bar{W}+\bar{X}+Y+\bar{Z})(\bar{W}+\bar{X}+Y+Z)$.

$$\begin{aligned} &= (\bar{W}+\bar{X}+\bar{Y})(\bar{W}+\bar{X}+Y+\bar{Z})(\bar{W}+\bar{X}+Y+Z) && \text{[T6(b)]}\\ &= (\bar{W}+\bar{X}+\bar{Y})(\bar{W}+\bar{X}+Y) && \text{[T6(b)]}\\ &= (\bar{W}+\bar{X}) && \text{[T6(b)]}\end{aligned}$$

Teorema 7.

(a) $ab + a\bar{b}c = ab + ac$.

(b) $(a+b)(a+\bar{b}+c) = (a+b)(a+c)$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} ab + a\bar{b}c &= a(b+\bar{b}c) && \text{[P5(b)]}\\ &= a(b+c) && \text{[T5(a)]}\\ &= ab + ac && \text{[P5(b)]}\end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 7 para simplificar las expresiones booleanas.

EJEMPLO 2.14

$$xy + x\bar{y}(\bar{w}+\bar{z}) = xy + x(\bar{w}+\bar{z}) \qquad \text{[T7(a)]}$$

EJEMPLO 2.15

$$(\bar{x}\bar{y}+z)(w+\bar{x}\bar{y}+\bar{z}) = (\bar{x}\bar{y}+z)(w+\bar{x}\bar{y}) \qquad \text{[T7(b)]}$$

EJEMPLO 2.16

$$\begin{aligned} (\bar{A}+\bar{B}+\bar{C})(\bar{B}+C)(A+\bar{B}) &= (\bar{A}+\bar{B})(\bar{B}+C)(A+\bar{B}) && \text{[T7(b)]}\\ &= \bar{B}(\bar{B}+C) && \text{[T6(b)]}\\ &= \bar{B} && \text{[T4(b)]}\end{aligned}$$

EJEMPLO 2.17

$$\begin{aligned} w\bar{y} + w\bar{x}y + wxyz + wx\bar{z} &= w\bar{y} + w\bar{x}y + wxy + wx\bar{z} && \text{[T7(a)]}\\ &= w\bar{y} + wy + wx\bar{z} && \text{[T6(a)]}\\ &= w + wx\bar{z} && \text{[T6(a)]}\\ &= w && \text{[T4(a)]}\end{aligned}$$

En los siguientes capítulos veremos que estos teoremas forman la base de algunos métodos estándar y automatizados por computadora para simplificar expresiones booleanas.

Al trabajar con álgebra booleana, con frecuencia necesitamos determinar el complemento de una expresión booleana. El siguiente teorema proporciona la base para esta operación.

Los siguientes tres teoremas son similares a la absorción en el sentido de que pueden servir para eliminar elementos adicionales de una expresión booleana.

Teorema 5.

(a) $a + \bar{a}b = a + b$.

(b) $a(\bar{a} + b) = ab$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} a + \bar{a}b &= (a + \bar{a})(a + b) && \text{[P5(a)]} \\ &= 1 \cdot (a + b) && \text{[P6(a)]} \\ &= (a + b) \cdot 1 && \text{[P3(b)]} \\ &= (a + b) && \text{[P2(b)]} \end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 5 para simplificar las expresiones booleanas.

EJEMPLO 2.7

$$B + A\bar{B}\bar{C}D = B + A\bar{C}D \qquad \text{[T5(a)]}$$

EJEMPLO 2.8

$$\bar{Y}(X + Y + Z) = \bar{Y}(X + Z) \qquad \text{[T5(b)]}$$

EJEMPLO 2.9

$$(X + Y)(\overline{(X + Y)} + Z) = (X + Y)Z \qquad \text{[T5(b)]}$$

EJEMPLO 2.10

$$AB + \overline{(AB)}C\bar{D} = AB + C\bar{D} \qquad \text{[T5(a)]}$$

Teorema 6.

(a) $ab + a\bar{b} = a$.

(b) $(a + b)(a + \bar{b}) = a$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} ab + a\bar{b} &= a(b + \bar{b}) && \text{[P5(b)]} \\ &= a \cdot 1 && \text{[P6(a)]} \\ &= a && \text{[P2(b)]} \end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 6 para simplificar expresiones booleanas.

EJEMPLO 2.11

$$ABC + A\bar{B}C = AC \qquad \text{[T6(a)]}$$

EJEMPLO 2.12

$$(AD + B + C)(AD + \overline{(B + C)}) = AD \qquad \text{[T6(b)]}$$

EJEMPLO 2.13

Simplificar $(\bar{W}+\bar{X}+\bar{Y}+\bar{Z})(\bar{W}+\bar{X}+\bar{Y}+Z)$
$(\bar{W}+\bar{X}+Y+\bar{Z})(\bar{W}+\bar{X}+Y+Z)$.

$$\begin{aligned} &= (\bar{W}+\bar{X}+\bar{Y})(\bar{W}+\bar{X}+Y+\bar{Z})(\bar{W}+\bar{X}+Y+Z) && \text{[T6(b)]}\\ &= (\bar{W}+\bar{X}+\bar{Y})(\bar{W}+\bar{X}+Y) && \text{[T6(b)]}\\ &= (\bar{W}+\bar{X}) && \text{[T6(b)]}\end{aligned}$$

Teorema 7.

(a) $ab + a\bar{b}c = ab + ac$.

(b) $(a+b)(a+\bar{b}+c) = (a+b)(a+c)$.

Demostración. La parte (a) del teorema se demuestra como sigue:

$$\begin{aligned} ab + a\bar{b}c &= a(b+\bar{b}c) && \text{[P5(b)]}\\ &= a(b+c) && \text{[T5(a)]}\\ &= ab+ac && \text{[P5(b)]}\end{aligned}$$

Los siguientes ejemplos ilustran el uso del teorema 7 para simplificar las expresiones booleanas.

EJEMPLO 2.14

$$xy + x\bar{y}(\bar{w}+\bar{z}) = xy + x(\bar{w}+\bar{z}) \qquad \text{[T7(a)]}$$

EJEMPLO 2.15

$$(\bar{x}\bar{y}+z)(w+\bar{x}\bar{y}+\bar{z}) = (\bar{x}\bar{y}+z)(w+\bar{x}\bar{y}) \qquad \text{[T7(b)]}$$

EJEMPLO 2.16

$$\begin{aligned} (\bar{A}+\bar{B}+\bar{C})(\bar{B}+C)(A+\bar{B}) &= (\bar{A}+\bar{B})(\bar{B}+C)(A+\bar{B}) && \text{[T7(b)]}\\ &= \bar{B}(\bar{B}+C) && \text{[T6(b)]}\\ &= \bar{B} && \text{[T4(b)]}\end{aligned}$$

EJEMPLO 2.17

$$\begin{aligned} w\bar{y} + w\bar{x}y + wxyz + wx\bar{z} &= w\bar{y} + w\bar{x}y + wxy + wx\bar{z} && \text{[T7(a)]}\\ &= w\bar{y} + wy + wx\bar{z} && \text{[T6(a)]}\\ &= w + wx\bar{z} && \text{[T6(a)]}\\ &= w && \text{[T4(a)]}\end{aligned}$$

En los siguientes capítulos veremos que estos teoremas forman la base de algunos métodos estándar y automatizados por computadora para simplificar expresiones booleanas.

Al trabajar con álgebra booleana, con frecuencia necesitamos determinar el complemento de una expresión booleana. El siguiente teorema proporciona la base para esta operación.

Teorema 8. Teorema de DeMorgan

(a) $\overline{a+b} = \bar{a} \cdot \bar{b}$.
(b) $\overline{a \cdot b} = \bar{a} + \bar{b}$.

Demostración. Demostremos la parte (a):

Si $X = a + b$, entonces $\overline{X} = (\overline{a+b})$. Por el postulado 6, $X \cdot \overline{X} = 0$ y $X + \overline{X} = 1$. Si $X \cdot Y = 0$ y $X + Y = 1$, entonces $Y = \overline{X}$, ya que el complemento de X es único. Por tanto, sea $Y = \bar{a}\bar{b}$ y calculemos $X \cdot Y$ y $X + Y$:

$$
\begin{aligned}
X \cdot Y &= (a+b)(\bar{a}\bar{b}) & \\
&= (\bar{a}\bar{b})(a+b) & [\text{P3(b)}] \\
&= (\bar{a}\bar{b})a + (\bar{a}\bar{b})b & [\text{P5(b)}] \\
&= a(\bar{a}\bar{b}) + (\bar{a}\bar{b})b & [\text{P3(b)}] \\
&= (a\bar{a})\bar{b} + \bar{a}(\bar{b}b) & [\text{P4(b)}] \\
&= 0 \cdot \bar{b} + \bar{a}(b \cdot \bar{b}) & [\text{P6(b), P3(b)}] \\
&= \bar{b} \cdot 0 + \bar{a} \cdot 0 & [\text{P3(b), P6(b)}] \\
&= 0 + 0 & [\text{T2(b)}] \\
&= 0 & [\text{P2(a)}]
\end{aligned}
$$

$$
\begin{aligned}
X + Y &= (a+b) + \bar{a}\bar{b} & \\
&= (b+a) + \bar{a}\bar{b} & [\text{P3(a)}] \\
&= b + (a + \bar{a}\bar{b}) & [\text{P4(a)}] \\
&= b + (a + \bar{b}) & [\text{T5(a)}] \\
&= (a + \bar{b}) + b & [\text{P3(a)}] \\
&= a + (\bar{b} + b) & [\text{P4(a)}] \\
&= a + (b + \bar{b}) & [\text{P3(a)}] \\
&= a + 1 & [\text{P6(a)}] \\
&= 1 & [\text{T2(a)}]
\end{aligned}
$$

Por tanto, por la unicidad de $\overline{X}$, $Y = \overline{X}$, y en consecuencia

$$\bar{a}\bar{b} = \overline{a+b}$$

Podemos generalizar el teorema 8 como sigue.

(a) $\overline{a + b + \cdots + z} = \bar{a} \cdot \bar{b} \cdot \cdots \cdot \bar{z}$.
(b) $\overline{ab \ldots z} = \bar{a} + \bar{b} + \cdots + \bar{z}$.

La regla para complementar una expresión es utilizar la relación (a) o (b), reemplazando cada operador + (OR) por un operador • (AND) y viceversa, y reemplazando cada variable por su complemento.

Debemos tener cuidado: al aplicar el teorema de DeMorgan, hay que respetar la precedencia de los operadores: • tiene precedencia sobre +. El siguiente ejemplo ilustra este punto importante.

EJEMPLO 2.18

Complementar la expresión $a + bc$.

$$\begin{aligned}\overline{a + b \cdot c} &= \overline{a + (b \cdot c)} \\ &= \bar{a} \cdot \overline{(b \cdot c)} \\ &= \bar{a} \cdot (\bar{b} + \bar{c}) \\ &= \bar{a}\bar{b} + \bar{a}\bar{c}\end{aligned}$$

Observe que: $\overline{a + b \cdot c} \neq \bar{a} \cdot \bar{b} + \bar{c}$

Los siguientes ejemplos ilustran el uso del teorema de DeMorgan.

EJEMPLO 2.19

$$\begin{aligned}\overline{X + \bar{Y}} &= \bar{X} \cdot \bar{\bar{Y}} && [\text{T8(a)}] \\ &= \bar{X} \cdot Y && [\text{T3}]\end{aligned}$$

EJEMPLO 2.20

Complementar la expresión $a\,(b + z(x + \bar{a}))$, y simplificar el resultado de modo que los únicos términos complementados sean variables individuales.

$$\begin{aligned}\overline{a(b + z(x + \bar{a}))} &= \bar{a} + \overline{(b + z(x + \bar{a}))} && [\text{T8(b)}] \\ &= \bar{a} + \bar{b}\,\overline{(z(x + \bar{a}))} && [\text{T8(a)}] \\ &= \bar{a} + \bar{b}(\bar{z} + \overline{(x + \bar{a})}) && [\text{T8(b)}] \\ &= \bar{a} + \bar{b}(\bar{z} + \bar{x} \cdot \bar{\bar{a}}) && [\text{T8(a)}] \\ &= \bar{a} + \bar{b}(\bar{z} + \bar{x}a) && [\text{T3}] \\ &= \bar{a} + \bar{b}(\bar{z} + \bar{x}) && [\text{T5(a)}]\end{aligned}$$

EJEMPLO 2.21

Repetir el ejemplo 2.20 para la expresión $a(b + c) + \bar{a}b$.

$$\begin{aligned}\overline{a(b + c) + \bar{a}b} &= \overline{ab + ac + \bar{a}b} && [\text{P5(b)}] \\ &= \overline{b + ac} && [\text{T6(a)}] \\ &= \bar{b}(\overline{ac}) && [\text{T8(a)}] \\ &= \bar{b}(\bar{a} + \bar{c}) && [\text{T8(b)}]\end{aligned}$$

Como ilustra este último ejemplo, con frecuencia podemos simplificar el proceso de complementar una expresión reduciéndola antes de aplicar el teorema de DeMorgan.

Así, el teorema de DeMorgan presenta la técnica general para complementar expresiones booleanas. Será de especial utilidad en la manipulación de expresiones booleanas a fin de darles formatos adecuados para su realización con tipos específicos de compuertas lógicas.

El último teorema fundamental del álgebra booleana que consideraremos es el teorema de consenso.

Teorema 9. Consenso

(a) $ab + \bar{a}c + bc = ab + \bar{a}c.$

(b) $(a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c).$

Demostración. De aquí en adelante, utilizaremos los postulados 3 y 4 sin indicarlo de manera explícita.

$$\begin{aligned} ab + \bar{a}c + bc &= ab + \bar{a}c + 1 \cdot bc && \text{[P2(b)]} \\ &= ab + \bar{a}c + (a + \bar{a})bc && \text{[P6(a)]} \\ &= ab + \bar{a}c + abc + \bar{a}bc && \text{[P5(b)]} \\ &= (ab + abc) + (\bar{a}c + \bar{a}cb) \\ &= ab + \bar{a}c && \text{[T4(a)]} \end{aligned}$$

La clave para utilizar este teorema es un elemento y su complemento, encontrar los términos asociados y eliminar el término incluido (el término de "consenso"), el cual está compuesto de los términos asociados.

El teorema de consenso es útil para reducir expresiones booleanas y para desarrollar expresiones en los diversos algoritmos de minimización automatizada que describiremos más adelante.

EJEMPLO 2.22

$$AB + \bar{A}CD + BCD = AB + \bar{A}CD \qquad \text{[T9(a)]}$$

EJEMPLO 2.23

$$(a + \bar{b})(\bar{a} + c)(\bar{b} + c) = (a + \bar{b})(\bar{a} + c) \qquad \text{[T9(b)]}$$

EJEMPLO 2.24

$$\begin{aligned} ABC + \bar{A}D + \bar{B}D + CD &= ABC + (\bar{A} + \bar{B})D + CD && \text{[P5(b)]} \\ &= ABC + \overline{AB}D + CD && \text{[T8(b)]} \\ &= ABC + \overline{AB}D && \text{[T9(a)]} \\ &= ABC + (\bar{A} + \bar{B})D && \text{[T8(b)]} \\ &= ABC + \bar{A}D + \bar{B}D && \text{[P5(b)]} \end{aligned}$$

En cada uno de los ejemplos anteriores, un elemento o expresión y su complemento son la clave para reducir la expresión.

Es importante señalar que es fácil demostrar los teoremas presentados por medio de diagramas de Venn. Por tanto, aconsejamos a los lectores utilizar esta imagen gráfica como ayuda para recordar los teoremas importantes. La tabla 2.2 resume los postulados y teoremas básicos del álgebra booleana. Más adelante presentaremos el teorema 10, incluido en esta tabla.

TABLA 2.2 POSTULADOS Y TEOREMAS DEL ÁLGEBRA BOOLEANA

Expresión	Dual
$P2(a): a+0=a$	$P2(b): a\cdot 1=a$
$P3(a): a+b=b+a$	$P3(b): ab=ba$
$P4(a): a+(b+c)=(a+b)+c$	$P4(b): a(bc)=(ab)c$
$P5(a): a+bc=(a+b)(a+c)$	$P5(b): a(b+c)=ab+ac$
$P6(a): a+\bar{a}=1$	$P6(b): a\cdot\bar{a}=0$
$T1(a): a+a=a$	$T1(b): a\cdot a=a$
$T2(a): a+1=1$	$T2(b): a\cdot 0=0$
$T3: \quad \bar{\bar{a}}=a$	
$T4(a): a+ab=a$	$T4(b): a(a+b)=a$
$T5(a): a+\bar{a}b=a+b$	$T5(b): a(\bar{a}+b)=ab$
$T6(a): ab+a\bar{b}=a$	$T6(b): (a+b)(a+\bar{b})=a$
$T7(a): ab+a\bar{b}c=ab+ac$	$T7(b): (a+b)(a+\bar{b}+c)=(a+b)(a+c)$
$T8(a): \overline{a+b}=\bar{a}\bar{b}$	$T8(b): \overline{ab}=\bar{a}+\bar{b}$
$T9(a): ab+\bar{a}c+bc=ab+\bar{a}c$	$T9(b): (a+b)(\bar{a}+c)(b+c)=(a+b)(\bar{a}+c)$

$T10(a): f(x_1, x_2, \ldots, x_n) = x_1 f(1, x_2, \ldots, x_n) + \bar{x}_1 f(0, x_2, \ldots, x_n)$

$T10(b): f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)][\bar{x}_1 + f(1, x_2, \ldots, x_n)]$

2.2 Funciones de conmutación

Hemos enunciado los postulados y teoremas del álgebra booleana en forma general, sin especificar los elementos del conjunto K. Por tanto, los resultados son válidos para cualquier álgebra booleana. En el siguiente análisis, nos centraremos en el álgebra booleana en la que $K=\{0, 1\}$. Esta formulación se conoce como *álgebra de conmutación*.

El concepto de función es bien conocido para los que están familiarizados con el álgebra ordinaria. Las funciones de conmutación representan el concepto correspondiente para el álgebra de conmutación y se pueden definir como sigue. Sean $X_1, X_2, \ldots, X_n$ símbolos llamados variables, cada uno de los cuales representa el elemento 0 o 1 de un álgebra de conmutación (se dice que 0 o 1 es el *valor* de la variable) y sea $f(X_1, X_2, \ldots, X_n)$ una función de conmutación de $X_1, X_2, \ldots, X_n$. La función f tiene el valor 0 o 1 según el conjunto de valores asignado a X_1, $X_2, \ldots, X_n$. Como hay n variables y cada variable tiene dos posibles valores, hay 2^n maneras de asignar estos valores a las n variables. Además, existen dos valores posibles para la función $f(x_1, x_2, \ldots, x_n)$. Por tanto, hay 2^{2^n} diferentes funciones de conmutación de n variables.

Si $n = 0$, las dos funciones de conmutación de cero variables son

$$f_0 = 0 \qquad f_1 = 1$$

Si $n = 1$, las cuatro funciones de la variable A son

$$f_0 = 0, \qquad f_2 = A$$
$$f_1 = \bar{A}, \qquad f_3 = 1$$

Deduciremos ahora las 16 funciones de dos variables A y B. Definamos $f_i(A, B)$ como sigue:

$$f_i(A, B) = i_3 AB + i_2 A\bar{B} + i_1 \bar{A}B + i_0 \bar{A}\bar{B}$$

donde $(i)_{10} = (i_3 i_2 i_1 i_0)_2$ adopta los valores binarios 0000, 0001, 0010, ..., 1111. Las 16 funciones resultantes son:

$$f_0(A, B) = 0$$
$$f_1(A, B) = \bar{A}\bar{B}$$
$$f_2(A, B) = \bar{A}B$$
$$f_3(A, B) = \bar{A}B + \bar{A}\bar{B} = \bar{A}$$
$$f_4(A, B) = A\bar{B}$$
$$f_5(A, B) = A\bar{B} + \bar{A}\bar{B} = \bar{B}$$
$$f_6(A, B) = A\bar{B} + \bar{A}B$$
$$f_7(A, B) = A\bar{B} + \bar{A}B + \bar{A}\bar{B} = \bar{A} + \bar{B}$$
$$f_8(A, B) = AB$$
$$f_9(A, B) = AB + \bar{A}\bar{B}$$
$$f_{10}(A, B) = AB + \bar{A}B = B$$
$$f_{11}(A, B) = AB + \bar{A}B + \bar{A}\bar{B} = \bar{A} + B$$
$$f_{12}(A, B) = AB + A\bar{B} = A$$
$$f_{13}(A, B) = AB + A\bar{B} + \bar{A}\bar{B} = A + \bar{B}$$
$$f_{14}(A, B) = AB + A\bar{B} + \bar{A}B = A + B$$
$$f_{15}(A, B) = AB + A\bar{B} + \bar{A}B + \bar{A}\bar{B} = 1$$

Al evaluar cada una de estas funciones para cada combinación de A y B, podemos resumir la información anterior en forma de tabla, como en la tabla 2.3.

TABLA 2.3 DIECISÉIS FUNCIONES DE DOS VARIABLES

AB	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
00	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
01	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
10	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
11	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Podemos describir una función de conmutación mediante una expresión de conmutación como sigue:

$$f(A, B, C) = AB + \bar{A}C + A\bar{C}$$

Si $A = 1$ y $B = C = 0$, el valor de la función f es 1, lo que verificamos como sigue:

$$\begin{aligned} f(1,0,0) &= 1 \cdot 0 + \bar{1} \cdot 0 + 1 \cdot \bar{0} \\ &= 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 && \text{[T3]} \\ &= 0 + 0 + 1 \cdot 1 && \text{[T2(b)]} \\ &= 1 \cdot 1 && \text{[P2(a)]} \\ &= 1 && \text{[P2(b)]} \end{aligned}$$

Podemos calcular otros valores de manera similar; por ejemplo, si $A = 0$, $B = 1$ y $C = 0$, podemos ver que $f = 0$.

2.2.1 Tablas de verdad

Podemos representar una función de conmutación mediante varias expresiones de conmutación diferentes, pero equivalentes. Si evaluamos una función de conmutación para todas las posibles combinaciones de entradas y presentamos los resultados como una tabla, obtenemos una representación única de la función llamada *tabla de verdad.*

Por ejemplo, podemos utilizar las tablas de verdad, como las tablas 2.4a, b y c, para mostrar las operaciones básicas OR, AND y de complemento utilizadas en el álgebra de conmutación, considerando cada una como una función de conmutación y exhibiendo todas las posibles combinaciones de los elementos.

Si evaluamos la función $f(A, B, C) = AB + \bar{A}C + A\bar{C}$ para todas las combinaciones de entradas posibles y las presentamos en forma de tabla, obtenemos la tabla de verdad que se muestra como tabla 2.5*a*. Al reemplazar cada 0 de la tabla 2.5*a* por *F* (falso) y cada 1 por *T* (verdadero) obtenemos una forma alternativa de la tabla de verdad, que se muestra en la tabla 2.5*b* y que

TABLA 2.4 TABLAS DE VERDAD PARA LAS FUNCIONES OR, AND Y NOT

$a\ b$	$f(a,b) = a + b$
0 0	0
0 1	1
1 0	1
1 1	1

(a)

$a\ b$	$f(a,b) = a \cdot b$
0 0	0
0 1	0
1 1	0
1 0	1

(b)

a	$f(a) = \bar{a}$
0	1
1	0

(c)

TABLA 2.5 TABLAS DE VERDAD PARA $f(A, B, C) = AB + \bar{A}C + A\bar{C}$

ABC	$f(A,B,C)$
000	0
001	1
010	0
011	1
100	1
101	0
110	1
111	1

(a)

ABC	$f(A,B,C)$
FFF	F
FFT	T
FTF	F
FTT	T
TFF	T
TFT	F
TTF	T
TTT	T

(b)

demuestra la correspondencia uno a uno que existe entre el álgebra de conmutación y el cálculo funcional de verdad [4].

También podemos utilizar la tabla de verdad como un medio conveniente para evaluar las funciones de conmutación. Por ejemplo, consideremos nuestra función anterior

$$f(A, B, C) = AB + \bar{A}C + A\bar{C}$$

Podemos obtener cada término de la tabla de verdad, uno a la vez, como sigue:

A, B, C	AB	$\bar{A}$	$\bar{A}C$	$AB+\bar{A}C$	$\bar{C}$	$A\bar{C}$	$(AB+\bar{A}C)+A\bar{C}$	$f(A, B, C)$
000	$0\cdot 0=0$	$\bar{0}=1$	$1\cdot 0=0$	$0+0=0$	$\bar{0}=1$	$0\cdot 1=0$	$0+0=0$	0
001	$0\cdot 0=0$	$\bar{0}=1$	$1\cdot 1=1$	$0+1=1$	$\bar{1}=0$	$0\cdot 0=0$	$1+0=1$	1
010	$0\cdot 1=0$	$\bar{0}=1$	$1\cdot 0=0$	$0+0=0$	$\bar{0}=1$	$0\cdot 1=0$	$0+0=0$	0
011	$0\cdot 1=0$	$\bar{0}=1$	$1\cdot 1=1$	$0+1=1$	$\bar{1}=0$	$0\cdot 0=0$	$1+0=1$	1
100	$1\cdot 0=0$	$\bar{1}=0$	$0\cdot 0=0$	$0+0=0$	$\bar{0}=1$	$1\cdot 1=1$	$0+1=1$	1
101	$1\cdot 0=0$	$\bar{1}=0$	$0\cdot 1=0$	$0+0=0$	$\bar{1}=0$	$1\cdot 0=0$	$0+0=0$	0
110	$1\cdot 1=1$	$\bar{1}=0$	$0\cdot 0=0$	$1+0=1$	$\bar{0}=1$	$1\cdot 1=1$	$1+1=1$	1
111	$1\cdot 1=1$	$\bar{1}=0$	$0\cdot 1=0$	$1+0=1$	$\bar{1}=0$	$1\cdot 0=0$	$1+0=1$	1

2.2.2 Formas algebraicas de las funciones de conmutación

Hasta ahora hemos visto varias formas diferentes de las funciones de conmutación, incluidas las expresiones algebraicas, las tablas de verdad y los diagramas de Venn. Ahora definiremos otras formas específicas de las funciones que demostrarán ser muy útiles.

Formas SOP y POS

Las funciones de conmutación en la forma de *suma de productos* (SOP) se construyen al sumar (OR) términos producto (AND), donde cada término producto se forma mediante el AND de varias variables complementadas o sin complementar, cada una de las cuales es una *literal*. Un ejemplo de forma SOP de una función de cuatro variables es

$$f(A, B, C, D) = A\bar{B}C + \bar{B}\bar{D} + \bar{A}C\bar{D}$$

Las funciones de conmutación en la forma de *producto de sumas* (POS) se construyen al considerar el producto (AND) de términos suma (OR), donde cada término suma se obtiene mediante el OR de varias literales. Un ejemplo de la forma POS de una función de cuatro variables es

$$f(A, B, C, D) = (\bar{A} + B + C)(\bar{B} + C + \bar{D})(A + \bar{C} + D)$$

Formas canónicas

Las formas canónicas de las funciones de conmutación son ciertas formas SOP y POS con características especiales. Como hemos visto, podemos representar una función de conmutación mediante expresiones de conmutación diferentes pero equivalentes. En cambio, las formas canónicas SOP y POS son únicas para cada función.

Mintérminos. Para una función de n variables, si un término producto contiene cada una de las n variables exactamente una vez, ya sea en forma complementada o no complementada, el término producto es un *mintérmino*. Si una función

representa como una suma sólo de mintérminos, decimos que la función tiene la forma de *suma canónica de productos* (*SOP canónica*). Por ejemplo,

$$f_\alpha(A, B, C) = \bar{A}B\bar{C} + AB\bar{C} + \bar{A}BC + ABC \tag{2.1}$$

es la forma SOP canónica de la función $f_\alpha(A, B, C)$, con cuatro mintérminos.

Para simplificar la escritura de la forma canónica SOP, con frecuencia se usa una notación especial, en la que cada mintérmino se representa mediante un código binario de n bits. Cada bit representa una de las variables del mintérmino como sigue:

Variable no complementada: 1
Variable complementada: 0

Las variables se enumeran en el mismo orden en cada mintérmino. Lo importante de esta notación es que, para que un mintérmino valga 1, cada variable no complementada del mintérmino debe valer 1 y cada variable complementada debe valer 0. Con este código, podemos escribir los mintérminos $f_\alpha(A, B, C)$ en una de las siguientes formas equivalentes:

Mintérmino	Código del mintérmino	Número de mintérmino
$\bar{A}B\bar{C}$	010	m_2
$AB\bar{C}$	110	m_6
$\bar{A}BC$	011	m_3
ABC	111	m_7

Escribimos cada mintérmino en forma abreviada como m_i, donde i es el entero decimal igual al código binario correspondiente para el mintérmino. Así, podemos escribir $f_\alpha(A, B, C)$ en forma compacta como

$$f_\alpha(A, B, C) = m_2 + m_3 + m_6 + m_7 \tag{2.2}$$

Podemos simplificar aún más si escribimos la función en *forma de lista de mintérminos* como sigue:

$$f_\alpha(A, B, C) = \sum m(2, 3, 6, 7) \tag{2.3}$$

Las tres ecuaciones (2.1), (2.2) y (2.3) ilustran tres formas diferentes, pero equivalentes, de representar la forma canónica SOP para $f_\alpha(A, B, C)$.

El orden de las variables en la notación funcional de las ecuaciones (2.2) y (2.3) es muy importante, ya que determina el orden de los bits en los números del mintérmino. Podemos demostrar este hecho fácilmente, cambiando la relación de orden de las variables en la función $f_\alpha(A, B, C)$ a $f_\alpha(B, C, A)$ como sigue:

$$\begin{aligned} f_\beta(B, C, A) &= \sum m(2, 3, 6, 7) \\ &= \underbrace{m_2}_{010} + \underbrace{m_3}_{011} + \underbrace{m_6}_{110} + \underbrace{m_7}_{111} \\ &= \bar{B}C\bar{A} + \bar{B}CA + BC\bar{A} + BCA \\ &= \bar{A}\bar{B}C + A\bar{B}C + \bar{A}BC + ABC \end{aligned} \tag{2.4}$$

Observe que la ecuación (2.4) no es idéntica a la (2.1), aunque las listas de mintérminos sean iguales. Continuamos trabajando con la ecuación (2.4) para obtener

$$\begin{aligned} f_\beta(A, B, C) &= f_\beta(B, C, A) \\ &= \underbrace{\bar{A}\bar{B}C}_{001} + \underbrace{\bar{A}BC}_{011} + \underbrace{A\bar{B}C}_{101} + \underbrace{ABC}_{111} \\ &= m_1 + m_3 + m_5 + m_7 \\ &= \sum m(1, 3, 5, 7) \end{aligned} \tag{2.5}$$

Las ecuaciones (2.4) y (2.5) son iguales; la diferencia en las listas de mintérminos refleja el orden de las variables en la notación funcional.

Podemos deducir fácilmente la tabla de verdad para $f_\beta(A, B, C)$ a partir de su forma SOP canónica:

Fila núm. (i)	**Entradas** ABC	m_1 $\bar{A}\bar{B}C$	m_3 $\bar{A}BC$	m_5 $A\bar{B}C$	m_7 ABC	**Salidas** $f_\beta(A, B, C)$
0	0 0 0	0	0	0	0	0
1	0 0 1	1	0	0	0	1
2	0 1 0	0	0	0	0	0
3	0 1 1	0	1	0	0	1
4	1 0 0	0	0	0	0	0
5	1 0 1	0	0	1	0	1
6	1 1 0	0	0	0	0	0
7	1 1 1	0	0	0	1	1

Un análisis cuidadoso de la tabla muestra que cada fila se numera según su código decimal, y que los únicos unos que aparecen en la tabla son los de las filas i, que corresponden a mintérminos m_i. Por tanto, en general, podemos eliminar todos los pasos intermedios y sólo escribir la tabla de verdad directamente a partir de la lista de mintérminos, como se muestra a continuación para la función $f_\alpha(A, B, C)$:

Fila núm. (i)	**Entradas** ABC	**Salidas** $f_\alpha(A, B, C)$	$= \sum m(2, 3, 6, 7)$	Complemento $\bar{f}_\alpha(A, B, C)$	$= \sum m(0, 1, 4, 5)$
0	0 0 0	0		1	$\leftarrow m_0$
1	0 0 1	0		1	$\leftarrow m_1$
2	0 1 0	1	$\leftarrow m_2$	0	
3	0 1 1	1	$\leftarrow m_3$	0	
4	1 0 0	0		1	$\leftarrow m_4$
5	1 0 1	0		1	$\leftarrow m_5$
6	1 1 0	1	$\leftarrow m_6$	0	
7	1 1 1	1	$\leftarrow m_7$	0	

Además, podemos ver que la tabla de verdad para $\bar{f}_\alpha(A, B, C)$ tiene unos en las filas 0, 1, 4 y 5. Por tanto,

$$f_\alpha(A, B, C) = \sum m(2, 3, 6, 7)$$

y

$$\bar{f}_\alpha(A, B, C) = \sum m(0, 1, 4, 5)$$

Observe que todos los mintérminos compuestos por tres variables (que suman $2^3 = 8$) están en la lista de mintérminos para $f_\alpha(A, B, C)$ o la lista para $\overline{f}_\alpha(A, B, C)$. En general, cada uno de los 2^n mintérminos de n variables siempre aparecerá en la forma SOP canónica para $f(x_1, x_2, ..., x_n)$ o la de $\overline{f}(x_1, x_2, ..., x_n)$.

EJEMPLO 2.25

Dada la función
$f(A,B,Q,Z) = \bar{A}\bar{B}\bar{Q}\bar{Z} + \bar{A}\bar{B}\bar{Q}Z + \bar{A}BQ\bar{Z} + \bar{A}BQZ,$
expresar las funciones $f(A, B, Q, Z)$ y $\overline{f}(A, B, Q, Z)$ en forma de lista de mintérminos.

$$\begin{aligned} f(A, B, Q, Z) &= \bar{A}\bar{B}\bar{Q}\bar{Z} + \bar{A}\bar{B}\bar{Q}Z + \bar{A}BQ\bar{Z} + \bar{A}BQZ \\ &= m_0 + m_1 + m_6 + m_7 \\ &= \sum m(0, 1, 6, 7) \end{aligned}$$

$\overline{f}(A, B, Q, Z)$ contendrá los 12 $(2^4 - 4)$ mintérminos restantes. La lista de mintérminos para esta función es

$$\begin{aligned} \bar{f}(A, B, Q, Z) &= m_2 + m_3 + m_4 + m_5 + m_8 + m_9 \\ &\quad + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15} \\ &= \sum m(2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15) \end{aligned}$$

Debemos recordar del álgebra de conmutación que

$$f(x_1, x_2, \ldots, x_n) + \bar{f}(x_1, x_2, \ldots, x_n) = 1$$

Sin embargo, como

$$f(x_1, x_2, \ldots, x_n) + \bar{f}(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{2^n-1} m_i$$

entonces

$$\sum_{i=0}^{2^n-1} m_i = 1 \tag{2.6}$$

En otras palabras, la suma (OR) de todos los mintérminos de n variables $(m_0, ..., m_{2^n-1})$ es igual a 1. Por último, es importante observar que, si bien

$$AB + \overline{AB} = 1 \qquad [P6(a)]$$

y

$$AB + \bar{A} + \bar{B} = 1 \qquad [T7(b)]$$

son expresiones válidas,

$$AB + \bar{A}\bar{B} \neq 1.$$

Un error común de los estudiantes del álgebra de conmutación es igualar la expresión anterior a 1.

Maxtérminos. Si un término suma de una función de n variables contiene cada una de las n variables exactamente una vez en forma complementada o no complementada, el término suma es un *maxtérmino*. Si representamos una

función como producto de términos suma, cada uno de los cuales es un maxtérmino, decimos que la función tiene la *forma canónica de producto de sumas* (*POS canónica*). Por ejemplo,

$$f_\gamma(A, B, C) = (A + B + C)(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C}) \quad (2.7)$$

es la forma POS canónica de una función $f_\gamma(A, B, C)$, que tiene cuatro maxtérminos.

Adoptamos una notación especial para los maxtérminos, como hicimos con los mintérminos, con la diferencia de que el código se intercambia como sigue:

Variable no complementada: 0
Variable complementada: 1

Lo importante de esta notación es que, para que un maxtérmino valga 0, toda variable no complementada del maxtérmino debe valer 0 y toda variable complementada debe valer 1. Así, podemos representar los maxtérminos de $f_\gamma(A, B, C)$ como sigue:

Maxtérmino	Código del maxtérmino	Lista de mintérmino
$A + B + C$	000	M_0
$A + B + \bar{C}$	001	M_1
$\bar{A} + B + C$	100	M_4
$\bar{A} + B + \bar{C}$	101	M_5

Escribimos cada maxtérmino en forma abreviada como M_i, donde i es el entero decimal del código binario correspondiente para el maxtérmino. Así,

$$f_\gamma(A, B, C) = M_0 M_1 M_4 M_5 \quad (2.8)$$

$$= \prod M(0, 1, 4, 5) \quad (2.9)$$

La última forma es la *forma de lista de maxtérminos*. Las ecuaciones (2.7), (2.8) y (2.9) son formas POS canónicas equivalentes para $f_\gamma(A, B, C)$. Como en el caso de las ecuaciones (2.2) y (2.3), el orden de las variables en las ecuaciones (2.8) y (2.9) es muy importante. La tabla de verdad para $f_\gamma(A, B, C)$ es

Núm. de fila (i)	**Entradas** ABC	M_0 $A + B + C$	M_1 $A + B + \bar{C}$	M_4 $\bar{A} + B + C$	M_5 $\bar{A} + B + \bar{C}$	**Salidas** $f_\gamma(A, B, C)$
0	000	0	1	1	1	0
1	001	1	0	1	1	0
2	010	1	1	1	1	1
3	011	1	1	1	1	1
4	100	1	1	0	1	0
5	101	1	1	1	0	0
6	110	1	1	1	1	1
7	111	1	1	1	1	1

Cada fila de la tabla se numera según el código decimal, como lo hicimos con los mintérminos. Observe que los únicos ceros de la tabla aparecen en los renglones i correspondientes a los maxtérminos M_i. Por tanto, como hicimos

con los mintérminos, podemos generar la tabla de verdad mediante una inspección directa de la lista de maxtérminos. Al comparar las tablas de verdad para $f_\alpha(A, B, C)$ y $f_\gamma(A, B, C)$ vemos que

$$\begin{aligned} f_\alpha(A, B, C) &= \sum m(2, 3, 6, 7) \\ &= f_\gamma(A, B, C) \\ &= \prod M(0, 1, 4, 5) \end{aligned} \tag{2.10}$$

Por tanto, las funciones $f_\alpha(A, B, C)$ y $f_\gamma(A, B, C)$ son iguales y la ecuación (2.10) muestra las formas SOP y POS canónicas para $f_\alpha(A, B, C)$.

EJEMPLO 2.26

Dada la función $f(A, B, C) = (A + B + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})$, construir la tabla de verdad y expresar la función con maxtérminos y con mintérminos.

$$\begin{aligned} f(A, B, C) &= \underbrace{(A + B + \bar{C})}_{001}\underbrace{(A + \bar{B} + \bar{C})}_{011}\underbrace{(\bar{A} + B + \bar{C})}_{101}\underbrace{(\bar{A} + \bar{B} + \bar{C})}_{111} \\ &= M_1 M_3 M_5 M_7 \\ &= \prod M(1, 3, 5, 7) \end{aligned}$$

Los maxtérminos colocan ceros en las filas 1, 3, 5 y 7 de la tabla de verdad.

Núm. de fila (i)	Entradas ABC	Salidas $f(A, B, C)$	$= \prod M(1, 3, 5, 7)$
0	0 0 0	1	
1	0 0 1	0	$\leftarrow M_1$
2	0 1 0	1	
3	0 1 1	0	$\leftarrow M_3$
4	1 0 0	1	
5	1 0 1	0	$\leftarrow M_5$
6	1 1 0	1	
7	1 1 1	0	$\leftarrow M_7$

De la tabla de verdad para $f(A, B, C)$, observamos que

$$f(A, B, C) = \sum m(0, 2, 4, 6)$$

Por tanto,

$$\begin{aligned} \bar{f}(A, B, C) &= \sum m(1, 3, 5, 7) \\ &= \underbrace{m_1}_{001} + \underbrace{m_3}_{011} + \underbrace{m_5}_{101} + \underbrace{m_7}_{111} \\ &= \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC \end{aligned}$$

En consecuencia,

$$\begin{aligned} f(A, B, C) &= \overline{\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC} \\ &= \overline{\bar{A}\bar{B}C} \cdot \overline{\bar{A}BC} \cdot \overline{A\bar{B}C} \cdot \overline{ABC} \end{aligned}$$

$$= \underbrace{(A + B + \bar{C})}_{001}\underbrace{(A + \bar{B} + \bar{C})}_{011}\underbrace{(\bar{A} + B + \bar{C})}_{101}\underbrace{(\bar{A} + \bar{B} + \bar{C})}_{111}$$
$$= M_1 M_3 M_5 M_7$$
$$= \prod M(1, 3, 5, 7)$$

Por tanto, hemos demostrado de manera algebraica que

$$f(A, B, C) = \prod M(1, 3, 5, 7) = \sum m(0, 2, 4, 6)$$

lo que es obvio si examinamos la tabla de verdad.

Al manejar funciones, queda clara una relación entre los mintérminos m_i y los maxtérminos M_i. Por ejemplo, para una función $f(A, B, C)$ tenemos que

$$\bar{m}_1 = \underbrace{\overline{\bar{A}\bar{B}C}}_{001} = \underbrace{A + B + \bar{C}}_{001} = M_1$$

(código del mintérmino) (código del maxtérmino)

y viceversa. Esto ocurre en el caso general; es decir,

$$\bar{m}_i = M_i \tag{2.11}$$
$$\bar{M}_i = \bar{\bar{m}}_i = m_i \tag{2.12}$$

Por tanto, los mintérminos y maxtérminos son complementarios entre sí.

EJEMPLO 2.27

Dada la función $f(A, B, C)$ del ejemplo 2.26, determinar la relación entre los maxtérminos para la función y su complemento.

La tabla de verdad es:

Núm. de fila (i)	Entradas ABC	Salidas $f(A, B, C)$	Salidas $\bar{f}(A, B, C)$	$= \prod M(0, 2, 4, 6)$
0	0 0 0	1	0	$\leftarrow M_0$
1	0 0 1	0	1	
2	0 1 0	1	0	$\leftarrow M_2$
3	0 1 1	0	1	
4	1 0 0	1	0	$\leftarrow M_4$
5	1 0 1	0	1	
6	1 1 0	1	0	$\leftarrow M_6$
7	1 1 1	0	1	

Como los renglones 0, 2, 4 y 6 tienen ceros, la forma canónica de $\bar{f}(A, B, C)$ es

$$\bar{f}(A, B, C) = \prod M(0, 2, 4, 6)$$

y, por tanto,

$$f(A, B, C) = \prod M(1, 3, 5, 7)$$

La función $f(A, B, C)$ tiene tres variables y, por tanto, ocho maxtérminos, todos los cuales aparecen en la lista de $f(A, B, C)$ o bien de $\bar{f}(A, B, C)$. Por el álgebra de conmutación, sabemos que

$$f(A, B, C) \cdot \bar{f}(A, B, C) = 0$$

y por tanto

$$(M_0 M_2 M_4 M_6)(M_1 M_3 M_5 M_7) = 0$$

o bien

$$\prod_{i=0}^{2^3-1} M_i = 0$$

Este ejemplo ilustra una relación que se cumple en general:

$$\prod_{i=0}^{2^n-1} M_i = 0 \tag{2.13}$$

Por último, observemos de la tabla de verdad que

$$f(A, B, C) = \sum m(0, 2, 4, 6) = \prod M(1, 3, 5, 7)$$

y

$$\bar{f}(A, B, C) = \sum m(1, 3, 5, 7) = \prod M(0, 2, 4, 6)$$

2.2.3 Deducción de formas canónicas

En los ejemplos anteriores mostramos el modo de traducir directamente las formas POS y SOP canónicas de una función en tablas de verdad y viceversa. Si expresamos una función en forma no canónica, con frecuencia es más conveniente utilizar el álgebra de conmutación para convertirla en su forma POS o SOP canónica, sin tener que deducir primero la tabla de verdad.

El siguiente teorema se utiliza con frecuencia en el desarrollo de expresiones de conmutación a una forma canónica.

Teorema 10. Teorema de desarrollo de Shannon

(a) $f(x_1, x_2, \ldots, x_n) = x_1 \cdot f(1, x_2, \ldots, x_n) + \bar{x}_1 \cdot f(0, x_2, \ldots, x_n)$

(b) $f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)][\bar{x}_1 + f(1, x_2, \ldots, x_n)]$

La base de este teorema la constituyen por el postulado 2 y el teorema 1. Como $x_1 = x_1 \bullet x_1 = x_1 \bullet 1$, podemos reemplazar cualquier x_1 dentro de la función de la parte (a) del teorema por 1, y de igual manera $x_1 = x_1 + x_1 = x_1 + 0$, lo que nos permite reemplazar cualquier x_1 dentro de la función de la parte (b) del teorema por 0. El teorema 10 es muy útil para desarrollar funciones o añadir literales a los términos producto.

EJEMPLO 2.28

Convertir la siguiente función de conmutación a la forma SOP canónica.

$$f(A, B, C) = AB + A\bar{C} + \bar{A}C$$

Aplicamos de manera sistemática el teorema 10a a esta función para las tres variables A, B y C. Para la variable A, el teorema 10a da

$$\begin{aligned} f(A, B, C) &= AB + A\bar{C} + \bar{A}C \\ &= A \cdot f(1, B, C) + \bar{A} \cdot f(0, B, C) \end{aligned}$$

$$= A(1 \cdot B + 1 \cdot \bar{C} + \bar{1} \cdot C) + \bar{A}(0 \cdot B + 0 \cdot \bar{C} + \bar{0} \cdot C)$$
$$= A(B + \bar{C}) + \bar{A}C$$

Para la variable B tenemos

$$\begin{aligned} f(A, B, C) &= A(B + \bar{C}) + \bar{A}C \\ &= B[A(1 + \bar{C}) + \bar{A}C] + \bar{B}[A(0 + \bar{C}) + \bar{A}C] \\ &= B[A + \bar{A}C] + \bar{B}[A\bar{C} + \bar{A}C] \\ &= AB + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}\bar{B}C \end{aligned}$$

Por último, para la variable C,

$$\begin{aligned} f(A, B, C) &= AB + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}\bar{B}C \\ &= C[AB + \bar{A}B \cdot 1 + A\bar{B} \cdot \bar{1} + \bar{A}\bar{B} \cdot 1] \\ &\quad + \bar{C}[AB + \bar{A}B \cdot 0 + A\bar{B} \cdot \bar{0} + \bar{A}\bar{B} \cdot 0] \\ &= ABC + \bar{A}BC + \bar{A}\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} \end{aligned}$$

Otro método para convertir expresiones a una forma SOP o POS canónica consiste en aplicar el teorema 6 para agregar literales a los términos producto o suma hasta obtener mintérminos o maxtérminos. Los siguientes ejemplos ilustran este procedimiento.

EJEMPLO 2.29

Convertir la siguiente función a forma SOP canónica:

$$f(A, B, C) = AB + A\bar{C} + \bar{A}C$$

Aplicamos el teorema 6a a cada uno de los tres términos producto de esta expresión.

$$AB = AB\bar{C} + ABC = m_6 + m_7$$
$$A\bar{C} = A\bar{C}\bar{B} + A\bar{C}B = A\bar{B}\bar{C} + AB\bar{C} = m_4 + m_6$$
$$\bar{A}C = \bar{A}C\bar{B} + \bar{A}CB = \bar{A}\bar{B}C + \bar{A}BC = m_1 + m_3$$

Por tanto,

$$\begin{aligned} f(A, B, C) &= AB + A\bar{C} + \bar{A}C \\ &= (m_6 + m_7) + (m_4 + m_6) + (m_1 + m_3) \\ &= \sum m(1, 3, 4, 6, 7) \end{aligned}$$

EJEMPLO 2.30

Desarrollar la siguiente función en forma POS canónica:

$$f(A, B, C) = A(A + \bar{C})$$

Podemos aplicar el teorema 6a como sigue para obtener los maxtérminos:

$$\begin{aligned} A &= (A + \bar{B})(A + B) \\ &= (A + \bar{B} + \bar{C})(A + \bar{B} + C)(A + B + \bar{C})(A + B + C) \\ &= M_3 M_2 M_1 M_0 \\ (A + \bar{C}) &= (A + \bar{C} + \bar{B})(A + \bar{C} + B) \\ &= (A + \bar{B} + \bar{C})(A + B + \bar{C}) \end{aligned}$$

$$= M_3 M_1$$

Por tanto,

$$A(A + C) = (M_3 M_2 M_1 M_0)(M_3 M_1)$$
$$= \prod M(0, 1, 2, 3)$$

2.2.4 Funciones con especificación incompleta

En el diseño de circuitos digitales, con frecuencia ocurre que la función de conmutación no tiene una especificación completa. En otras palabras, puede ocurrir que una función deba contener ciertos mintérminos y omitir otros, y que los mintérminos restantes sean opcionales. En este caso, podemos incluir los mintérminos opcionales en el diseño lógico si éstos ayudan a simplificar el circuito lógico, o bien omitirlos. Un mintérmino opcional es un *mintérmino prescindible*. Si expresamos una función mediante sus maxtérminos, por lo general escribimos los mintérminos prescindibles en la forma correspondiente de maxtérminos, llámandose en este caso *maxtérminos prescindibles*.

Los términos prescindibles surgen de dos formas. En la primera, podría ocurrir que ciertas combinaciones de entrada nunca se aplican a una red de conmutación en particular; por tanto, como nunca aparecen, podemos utilizar sus mintérminos a nuestro antojo. Estas condiciones prescindibles ocurren de manera natural en muchas aplicaciones prácticas. Por ejemplo, suponga que una red de conmutación tiene entradas $a_3a_2a_1a_0$, las cuales representan dígitos del código binario decimal (BCD) definido en la tabla 2.6. (Recuerde que analizamos el código BCD en el capítulo 1.) Sólo pueden aparecer 10 mintérminos, $m_0 \ldots m_9$, los correspondientes a los 10 dígitos decimales. Los otros seis mintérminos, $m_{10} \ldots m_{15}$, no pueden aparecer y, por tanto, son prescindibles en todas las situaciones. En consecuencia, podemos incluir u omitir estos términos en las expresiones de conmutación para cualquier función $f(a_3a_2a_1a_0)$ de estas entradas.

TABLA 2.6 CÓDIGO DECIMAL CODIFICADO EN BINARIO (BCD)

Dígito decimal	Código BCD $a_3a_2a_1a_0$	Dígito decimal	Código BCD $a_3a_2a_1a_0$
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Las condiciones prescindibles también surgen cuando ocurren todas las combinaciones de entradas de una red dada, pero sólo se requiere que la salida sea 1 o 0 para ciertas combinaciones.

Al escribir las expresiones de conmutación, indicaremos los mintérminos prescindibles como d_i en vez de m_i, y los maxtérminos prescindibles como D_i en vez de M_i, como en el siguiente ejemplo.

EJEMPLO 2.31

Sea $f(A, B, C)$ una función con mintérminos m_0, m_3 y m_7 y condiciones prescindibles d_4 y d_5. Expresar la función y su complemento con mintérminos y con maxtérminos; reducir después la función a su forma más sencilla.

La forma de lista de mintérminos para esta función es

$$f(A, B, C) = \sum m(0, 3, 7) + d(4, 5)$$

y la lista de maxtérminos es

$$f(A, B, C) = \prod M(1, 2, 6) \cdot D(4, 5)$$

Observe que los maxtérminos prescindibles D_i son sencillamente los mintérminos prescindibles, ya que los términos pueden ser 1 o 0. De aquí que

$$\begin{aligned} \bar{f}(A, B, C) &= \sum m(1, 2, 6) + d(4, 5) \\ &= \prod M(0, 3, 7) \cdot D(4, 5) \end{aligned}$$

Para simplificar la expresión $f(A, B, C)$, enumeramos los términos como

$$f(A, B, C) = \bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC + d(A\bar{B}\bar{C} + A\bar{B}C)$$

Observe que el segundo y tercer término difieren sólo en una literal y, por tanto, podemos agruparlos para obtener

$$f(A, B, C) = \bar{A}\bar{B}\bar{C} + BC + d(A\bar{B}\bar{C} + A\bar{B}C)$$

Sin el uso de los prescindibles, no podríamos simplificar aún más la función. Sin embargo, recordemos que los prescindibles, por definición, pueden ser cero o uno. Por tanto, podemos utilizarlos u omitirlos, según ayuden o no en la simplificación. En la función anterior, si optamos por utilizar d_4 y omitir d_5, la función se convierte en

$$\begin{aligned} f(A, B, C) &= \bar{A}\bar{B}\bar{C} + BC + A\bar{B}\bar{C} \\ &= \bar{B}\bar{C} + BC \end{aligned}$$

que es la forma más sencilla de la función. Podemos realizar un análisis similar con la función en forma de maxtérminos.

2.3 Circuitos de conmutación

Los circuitos lógicos digitales, o *circuitos de conmutación*, como se les llama con frecuencia, constan de combinaciones seriales y paralelas de elementos de conmutación llamados *compuertas*, o se implantan mediante arreglos lógicos programables o dispositivos similares. Desde un punto de vista matemático, las compuertas son sólo rutas de señales abiertas o cerradas. Desde un punto de vista tecnológico, las compuertas son dispositivos electrónicos de conmutación de gran velocidad que pueden activarse o desactivarse en pocos nanosegundos. En este capítulo analizamos el uso de compuertas para la construcción de circuitos lógicos que realicen funciones de conmutación. Analizaremos el diseño con arreglos lógicos programables en el capítulo 6.

2.3.1 Compuertas lógicas electrónicas

En los circuitos lógicos digitales, podemos asociar las variables de conmutación a las condiciones de entrada de las compuertas; es decir, a niveles de voltaje altos o bajos aplicados a las entradas de una compuerta. Las funciones de conmutación pueden corresponder a la salida de una compuerta o sistema de compuertas, representadas por un nivel de voltaje alto o bajo en la salida.

Señales eléctricas y valores lógicos

En los libros de datos, las tablas de verdad que definen la operación de los circuitos de compuertas lógicas se presentan en términos de un voltaje alto (H) y otro bajo (L). El diseñador puede utilizar estos niveles de voltaje para representar los valores lógicos 0 y 1 de diversas formas. La convención de *lógica positiva* utiliza un voltaje alto (*H*) para representar el 1 lógico y un voltaje bajo (*L*) para representar el 0 lógico. La convención de *lógica negativa* utiliza *L* para representar el 1 lógico y *H* para el 0 lógico. En un sistema de *lógica mixta*, se utiliza la lógica positiva para algunas señales y la negativa para otras.

Una señal de 1 lógico es *afirmada*, *activa* o *verdadera*. Una señal *alta activa* se afirma cuando es alta (en lógica positiva), mientras que una señal *baja activa* se afirma cuando es baja (en lógica negativa). Si una señal está afirmada, es decir, si indica 0 lógico, es una señal *no afirmada*, *negada* o *falsa*. Utilizamos el término *polaridad* para referirnos a la naturaleza alta activa o baja activa de una señal lógica.

Al representar las señales mediante variables lógicas, escribimos los nombres de la señal baja activa en forma complementada (por ejemplo, $\overline{a}$, a', a^*), y los de la señal alta activa en forma no complementada (a). Debemos elegir cada nombre de señal de modo que el nombre sugiera el propósito de ésta. Por ejemplo, el nombre de señal *RUN* sugiere una señal afirmada (alta) para que comience a trabajar un equipo. Si la señal es baja activa, deberemos utilizar el nombre de señal $\overline{RUN}$ para indicar que el equipo trabajará cuando la señal esté afirmada baja. En este texto utilizaremos principalmente señales altas activas, pero como muchos módulos de circuitos comerciales tienen entradas o salidas bajas activas, también presentaremos varios ejemplos en los que se utilicen señales bajas activas.

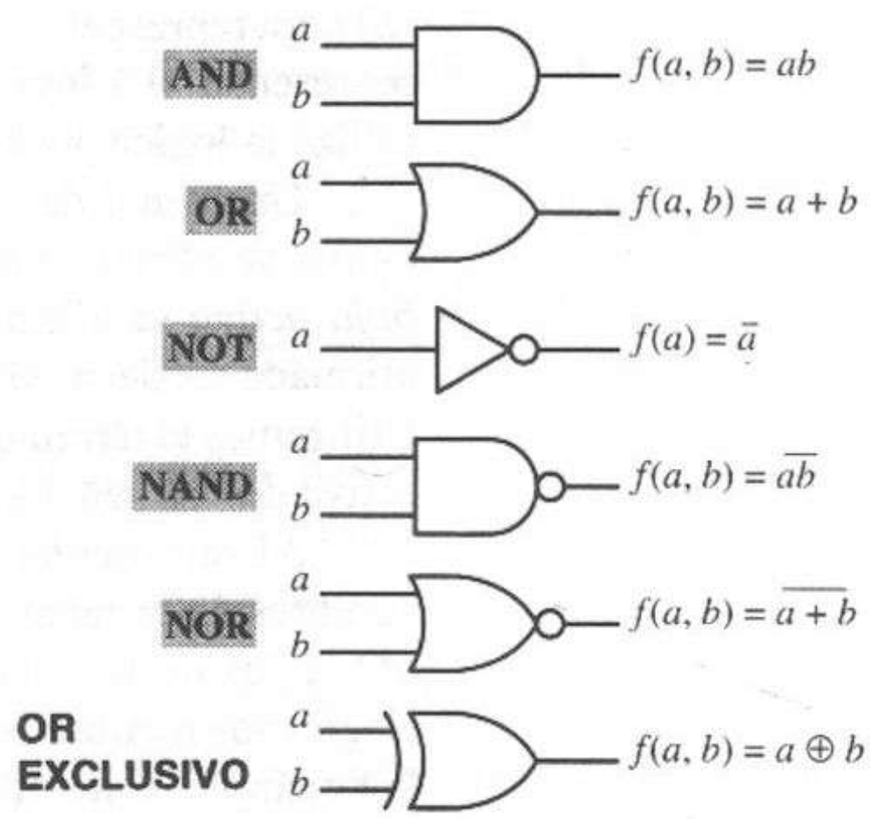
Símbolos de compuerta

Representamos cada compuerta en un diagrama de circuitos lógicos mediante un símbolo que incluye las entradas y las salidas. El número de entradas de una compuerta se conoce como su *fan-in* (ábanico de entrada). Hay módulos de circuitos estándar que contienen compuertas AND, OR, NAND y NOR con un número limitado de opciones de *fan-in*; por lo general, las compuertas tienen dos, tres, cuatro u ocho entradas. Los dispositivos lógicos programables y los circuitos adaptados proporcionan por lo general una amplia gama de opciones *fan-in*, lo que hace posible una correspondencia más directa entre el circuito y la expresión lógica por realizar.

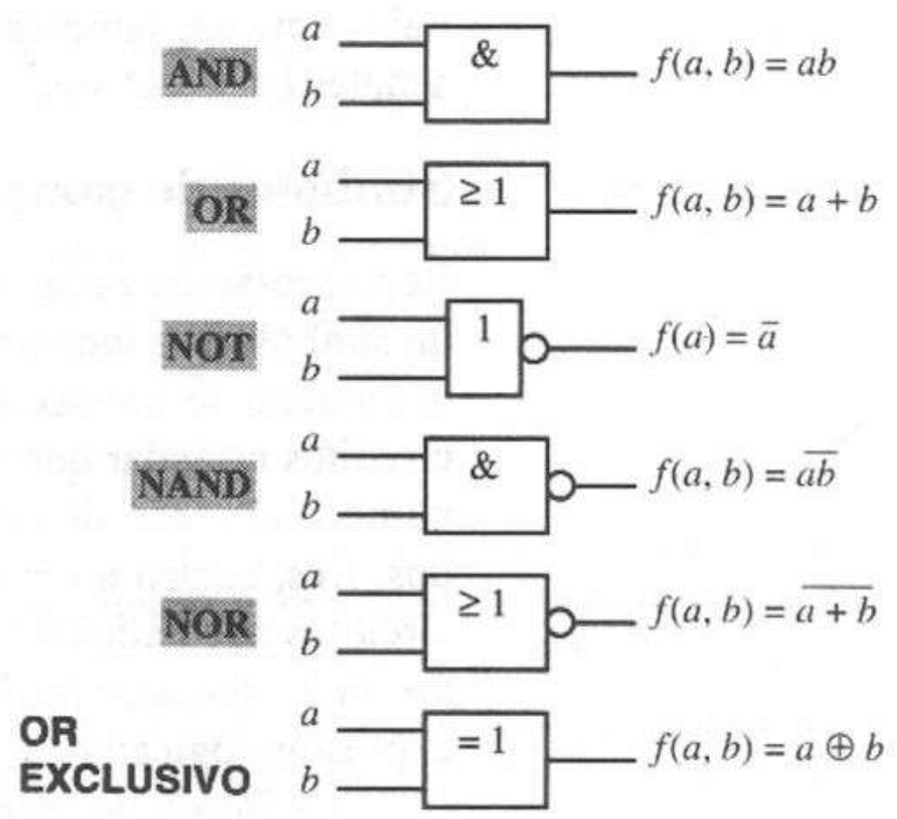
La forma del cuerpo del símbolo representa la función lógica básica, u operación booleana, realizada por la compuerta (OR, AND, NOT u otras). Las *burbujas* dibujadas en las entradas o salidas de un símbolo lógico indican señales bajas activas. Una burbuja en una entrada indica que la entrada es baja activa, es decir, que debe estar afirmada baja para obtener un 1 lógico como entrada de la función. La ausencia de burbuja indica una entrada alta activa; la entrada se

afirma con el valor 1 lógico. De igual manera, una burbuja en una salida indica una salida baja activa, lo que implica que si la evaluación de la función produce 1, se obtiene un 0 lógico en la salida.

Para entender los circuitos lógicos, debemos dominar los operadores de la figura 2.4. Los conjuntos de símbolos ilustrados se definen en la norma IEEE/ ANSI Estándar 91-1984 [7]. Analizaremos cada uno de estos operadores, o *compuertas*, en la siguiente sección. En este texto, cuando dibujemos circuitos compuestos por compuertas discretas, utilizaremos principalmente el conjunto de símbolos 1, en el que se identifican las funciones lógicas por medio de formas distintivas de los símbolos. Utilizaremos el conjunto de símbolos 2 en capítulos posteriores para describir módulos funcionales mayores.



Conjunto de símbolos 1



Conjunto de símbolos 2 (Estándar ANSI/IEEE 91-1984)

Figura 2.4 Símbolos para dispositivos de conmutación.

Los módulos de circuitos disponibles en el mercado que contienen configuraciones específicas de compuertas lógicas discretas aparecen en la figura 2.5. Estos módulos se utilizan para construir circuitos útiles para aplicaciones prácticas. Los grupos de compuertas individuales implantados en un módulo lógico se caracterizan como módulos con integración a pequeña escala (SSI) y contienen entre 10 y 100 transistores que conforman el módulo total. Los módulos descritos están disponibles en paquetes duales en línea (DIP) con las asignaciones

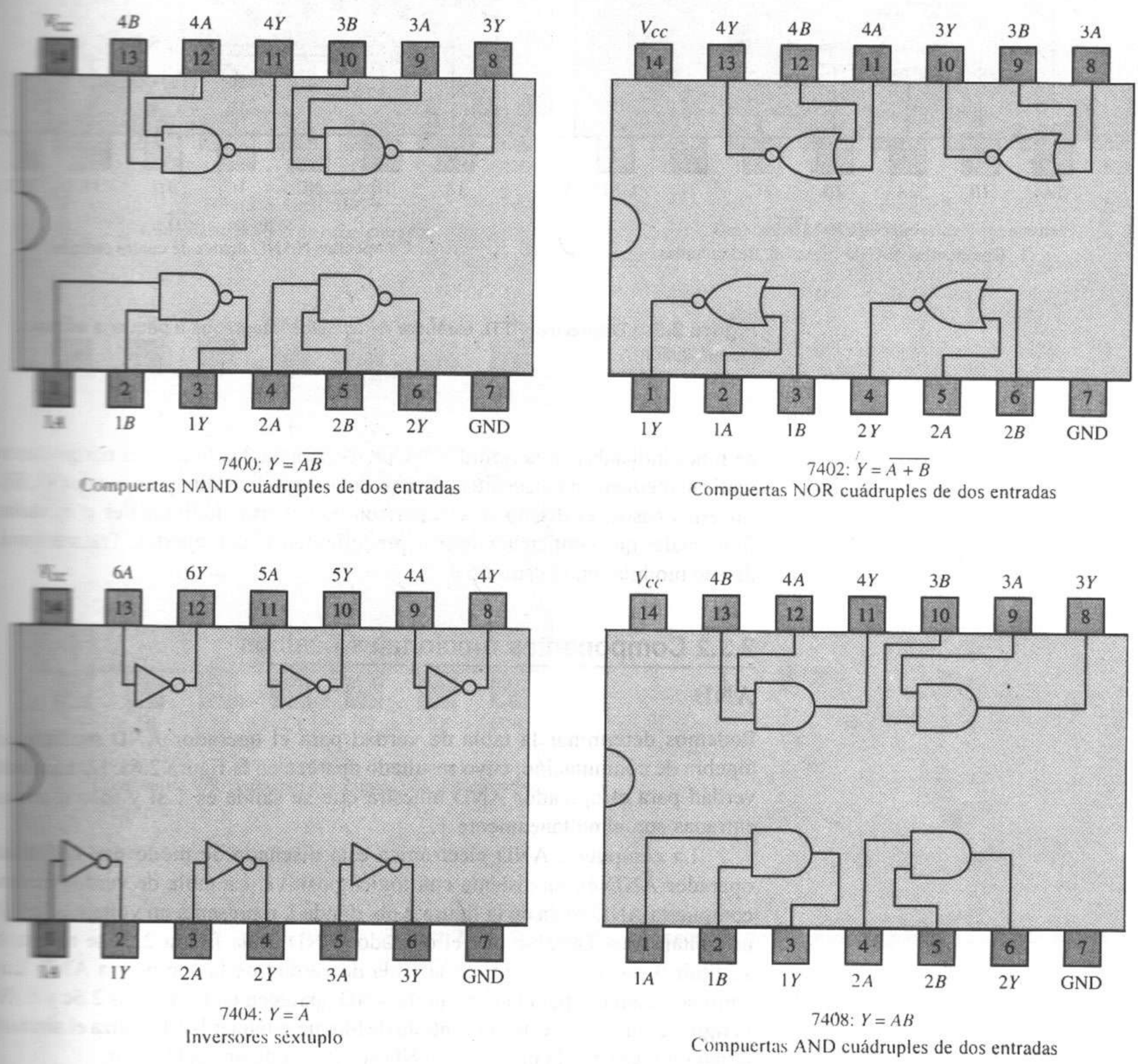
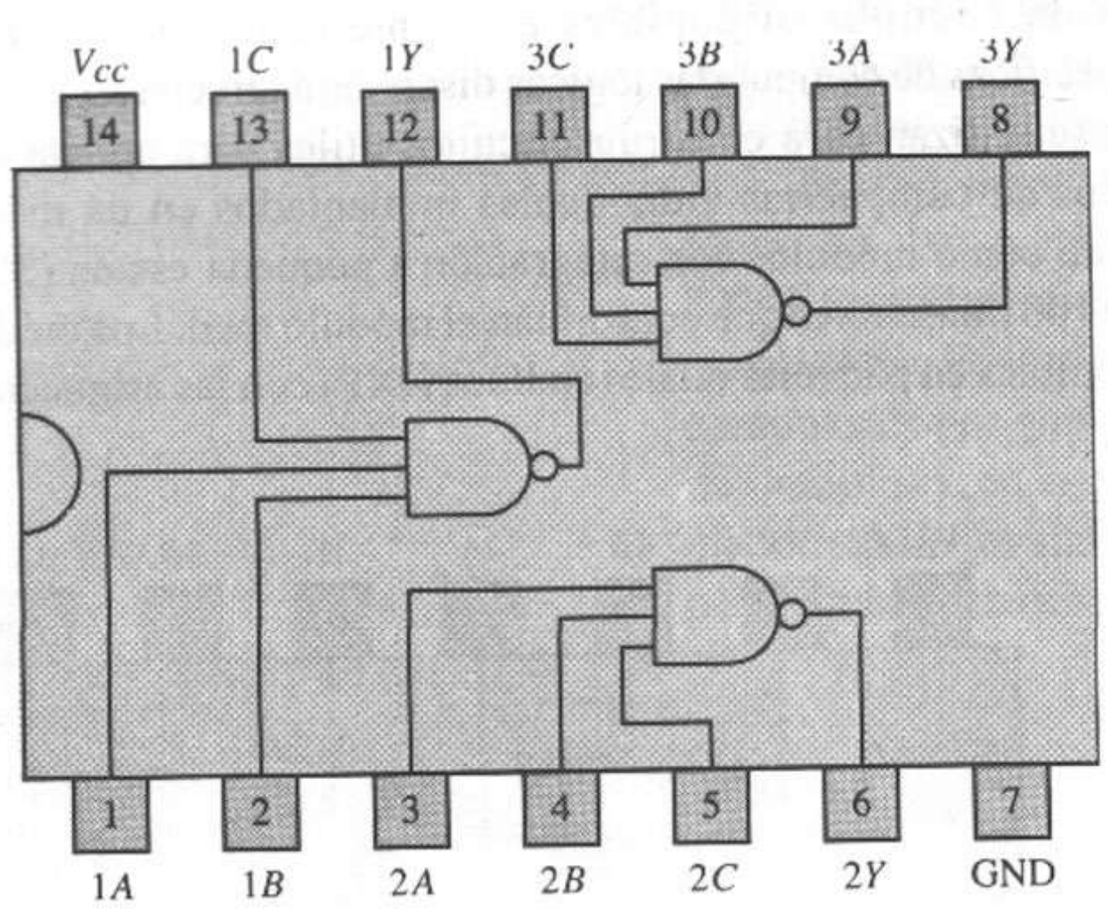
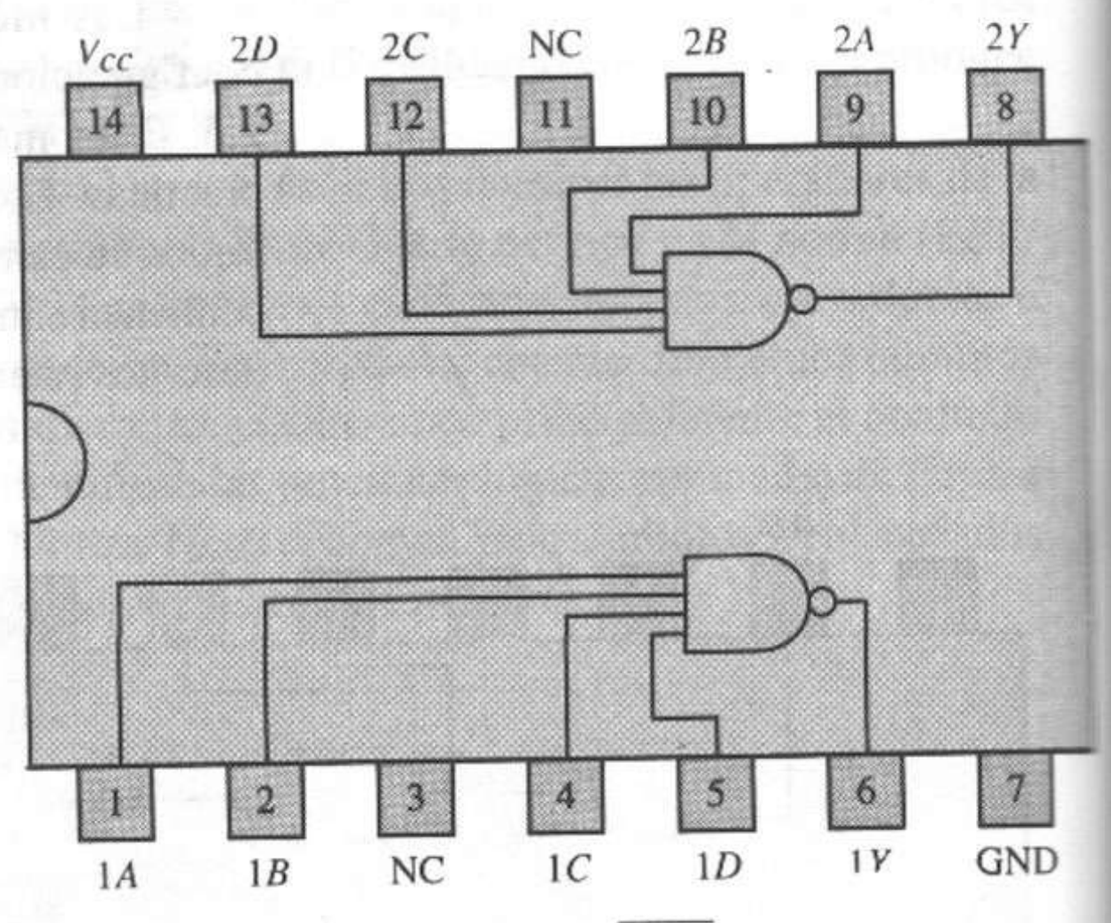


Figura 2.5 Dispositivos TTL estándar de circuitos integrados a pequeña escala (vista superior).



7410: $Y = \overline{ABC}$
Compuertas NAND triples de tres entradas



7420: $Y = \overline{ABCD}$
Compuertas NAND duales de cuatro entradas

Figura 2.5 Dispositivos TTL estándar de circuitos integrados a pequeña escala (continuación).

de pines indicadas en la figura 2.5. Con frecuencia, las funciones completas se realizan mediante un dispositivo de circuito integrado a muy gran escala (VLSI). En estos casos, el diseño se efectúa con compuertas individuales o módulos funcionales que contienen patrones predefinidos de compuertas. Trataremos el diseño modular en el capítulo 4.

2.3.2 Componentes funcionales básicos

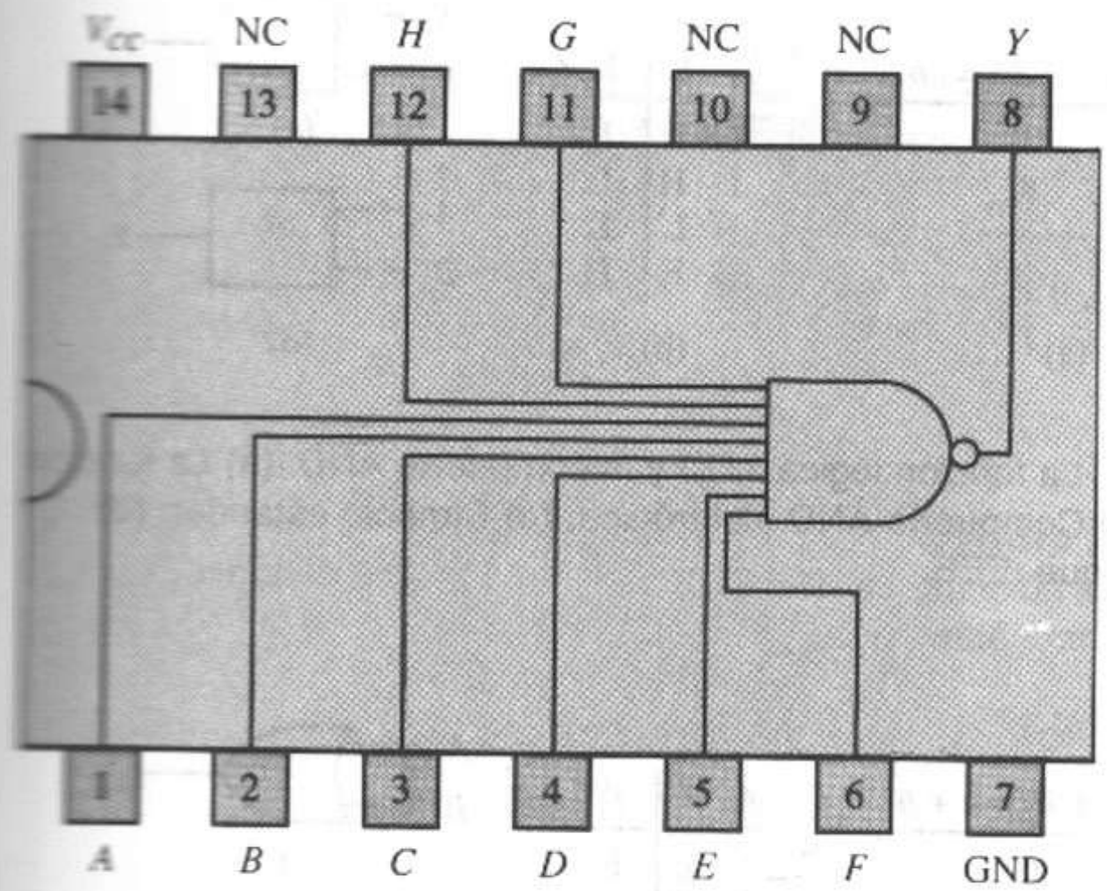
AND

Podemos determinar la tabla de verdad para el operador AND mediante el álgebra de conmutación, cuyo resultado aparece en la figura 2.6a. Esta tabla de verdad para el operador AND muestra que su salida es 1 si y sólo si ambas entradas son simultáneamente 1.

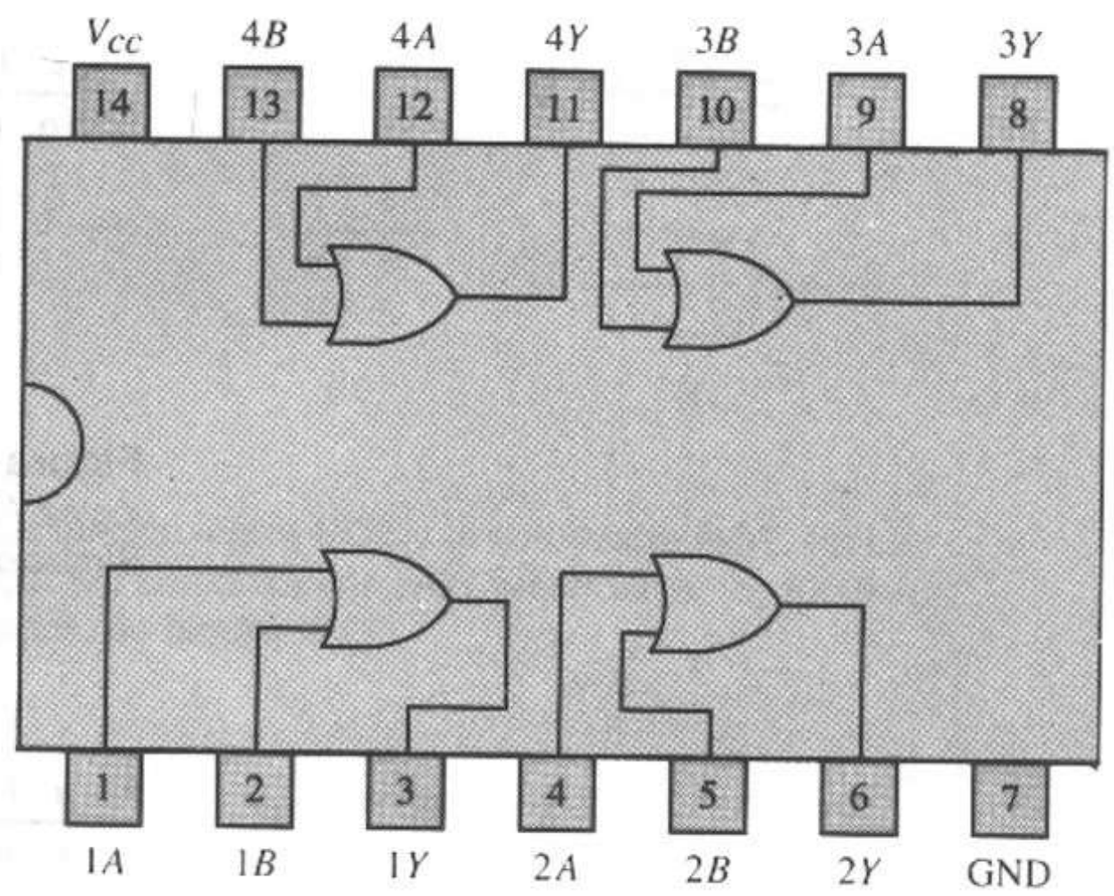
La compuerta AND electrónica está diseñada de modo que realice el operador AND en un sistema con lógica positiva. La tabla de verdad de una compuerta AND se da en la figura 2.6b, donde L representa un voltaje bajo y H un voltaje alto. Observe que el operador AND de la figura 2.6a se realiza al sustituir 0 por L y 1 por H en la tabla de verdad de la compuerta AND. Los símbolos estándar para la compuerta AND aparecen en las figuras 2.6c y d. En la figura 2.6d, observe que el símbolo de bloque estándar IEEE utiliza el símbolo & para indicar que la operación AND se ejecuta dentro del bloque.

OR

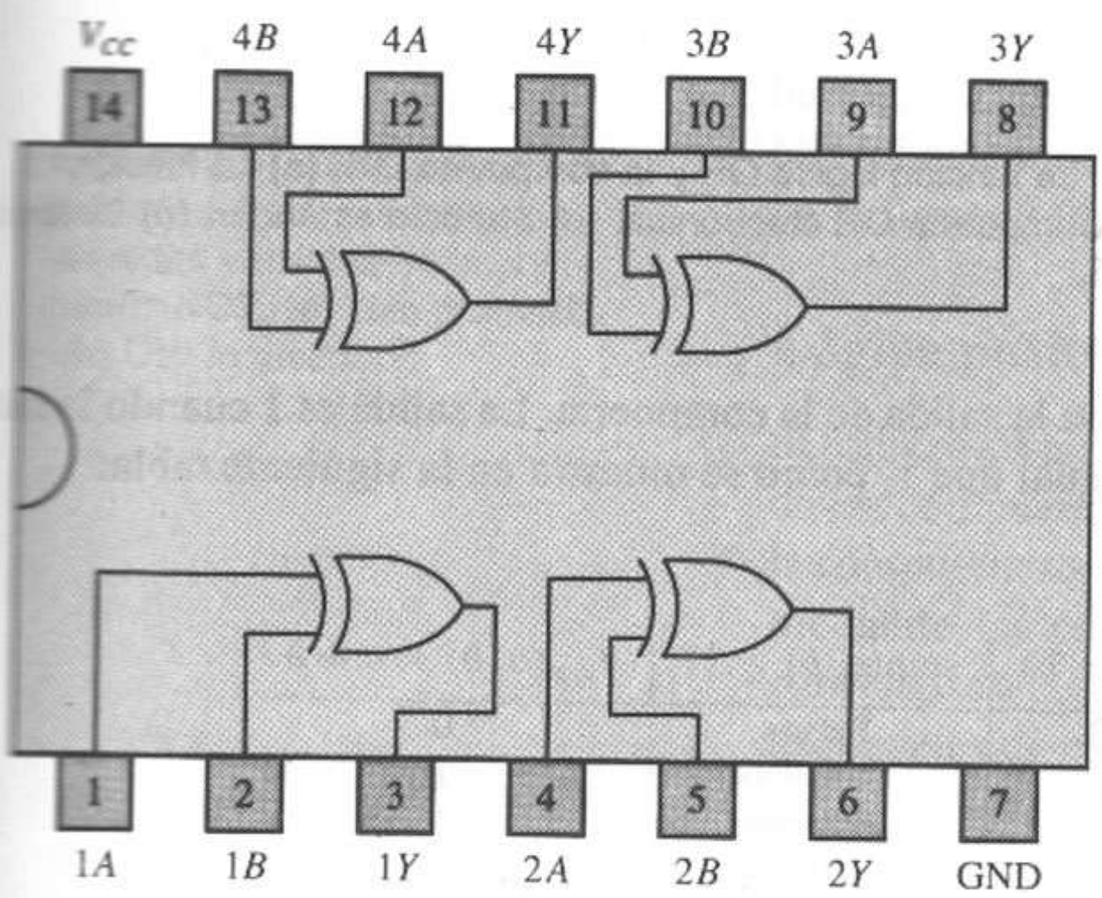
La función OR es idéntica al operador OR del álgebra de conmutación; su tabla de verdad aparece en la figura 2.7a. Observe que la salida es 0 si y sólo si ambas entradas son 0, y 1 si una o más entradas son 1. La tabla de verdad correspondiente



7430: $Y = \overline{ABCDEFGH}$
Compuerta NAND de ocho entradas



7432: $Y = A + B$
Compuertas OR cuádruples de dos entradas



7486: $Y = A \oplus B$
Compuertas OR exclusivo cuádruples con dos entradas

Figura 2.5 Dispositivos TTL estándar de circuitos integrados a pequeña escala (continuación).

de una compuerta OR electrónica se da en la figura 2.7b. Cabe señalar que la compuerta OR realiza el operador OR en un sistema con lógica positiva. Los símbolos estándar de la compuerta OR aparecen en las figuras 2.7c y d. En la figura 2.7d, observe que el símbolo de bloque de IEEE contiene la designación ≥ 1. Esto significa que la suma matemática de los valores de las variables de

a	b	$f_{AND}(a, b) = ab$
0	0	0
0	1	0
1	0	0
1	1	1

(a)

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

(b)

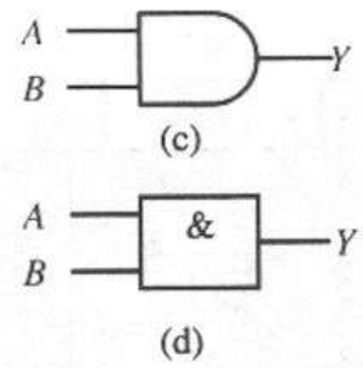


Figura 2.6 La función lógica AND y la compuerta AND. **(a)** La función lógica AND. **(b)** Compuerta AND electrónica. **(c)** Símbolo estándar. **(d)** Símbolo de bloque IEEE.

a	b	$f_{OR}(a, b) = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

(a)

A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

(b)

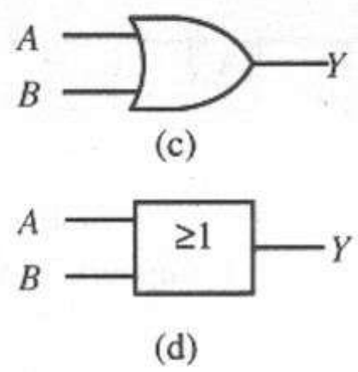


Figura 2.7 La función lógica OR y la compuerta OR. **(a)** La función lógica OR. **(b)** Compuerta OR electrónica. **(c)** Símbolo estándar. **(d)** Símbolo de bloque IEEE.

entrada a y b determina la salida de la compuerta. La salida es 1 cuando la suma de a y b es mayor o igual que 1, como se muestra en la siguiente tabla:

$a\ b$	sum(a, b)	sum$(a, b) \geq 1$?	$f_{OR}(a, b) = a + b$
0 0	0	Falso	0
0 1	1	Verdadero	1
1 0	1	Verdadero	1
1 1	2	Verdadero	1

NOT

Una compuerta NOT (Fig. 2.8), o *inversor*, siempre tiene exactamente una entrada y se utiliza para implantar el concepto de complemento del álgebra de conmutación. Cualquier variable tiene sus formas verdadera (no complementada) y falsa (complementada), a y $\overline{a}$, respectivamente. Utilizamos una compuerta NOT para obtener una a partir de la otra.

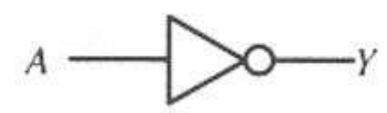
Los símbolos estándar para la compuerta NOT, que se muestran en las figuras 2.8c y d, incluyen una burbuja en la salida de la compuerta. Como ya hemos señalado, una burbuja en la salida de cualquier elemento de circuito lógico indica que un 1 lógico interno produce un 0 lógico externo y, de manera similar, un 0 lógico interno produce un 1 lógico externo. La compuerta NOT no realiza ninguna otra función lógica; por tanto, el valor lógico de la salida de una compuerta NOT es sólo el complemento del valor lógico de su entrada.

a	$f_{NOT}(a) = \bar{a}$
0	1
1	0

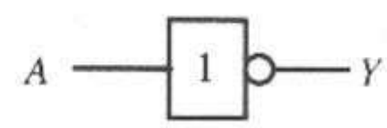
(a)

A	Y
L	H
H	L

(b)



(c)



(d)

Figura 2.8 La función lógica NOT y la compuerta NOT. **(a)** La función lógica NOT. **(b)** La compuerta NOT electrónica. **(c)** Símbolo estándar. **(d)** Símbolo de bloque IEEE.

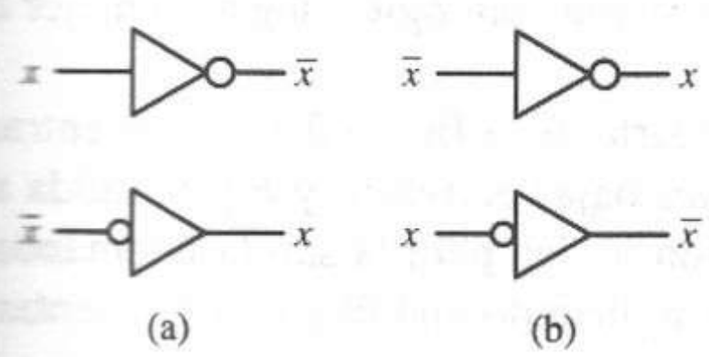


Figura 2.9 Concordancia de polaridad de la señal con las entradas y salidas de la compuerta NOT. **(a)** Uso preferido. **(b)** Uso inadecuado.

Podemos visualizar una compuerta NOT como un cambio de polaridad de una señal alta activa a baja activa, o viceversa. En consecuencia, podemos dibujar el símbolo de la compuerta NOT con la burbuja en la entrada o en la salida. Por convención, dibujamos la burbuja en la entrada de la compuerta cuando la señal de entrada es baja activa, y en la salida de la compuerta si la señal emitida es baja activa. La figura 2.9a muestra el uso recomendado del símbolo de compuerta NOT, haciendo concordar las burbujas con la señal baja activa, $\bar{x}$. Los diagramas de la figura 2.9b, aunque no son incorrectos, no se recomiendan.

Lógica positiva contra lógica negativa

Si utilizamos la convención de la lógica positiva para todas las entradas y salidas de las compuertas, es decir, si las señales conectadas a las entradas y salidas de la compuerta son todas altas activas, las funciones lógicas AND y OR se realizan mediante las compuertas AND y OR, respectivamente. Cuando las señales conectadas a las entradas y salidas de la compuerta son bajas activas, se invierten los papeles de estas compuertas.

Recuerde que en la convención de lógica negativa, el 1 se representa con un voltaje bajo y el 0 con un voltaje alto. Por tanto, podemos deducir la función realizada por una compuerta AND en el sistema de lógica negativa al sustituir 0 por H y 1 por L en la tabla de verdad de la compuerta AND de la figura 2.6b. La tabla resultante, que aparece en la figura 2.10a, es idéntica a la tabla de verdad del operador OR de la figura 2.7a. Así, podemos considerar que una compuerta AND con entradas y salidas bajas activas realiza la función lógica OR.

Podemos verificar esto con el álgebra de conmutación si aplicamos la involución (teorema 3) y el teorema de DeMorgan (teorema 8) a la expresión de la función lógica AND, como sigue:

$$\begin{aligned} y &= a \cdot b \\ &= \overline{\overline{a \cdot b}} \\ &= \overline{\bar{a} + \bar{b}} \\ &= \bar{f}_{OR}(\bar{a}, \bar{b}) \end{aligned} \tag{2.14}$$

La ecuación 2.14 indica que podemos dibujar un símbolo de compuerta AND como una función OR con entradas y salida bajas activas, como se muestra en la figura 2.10b. Aunque al principio parezca extraño, este símbolo ilustra la función

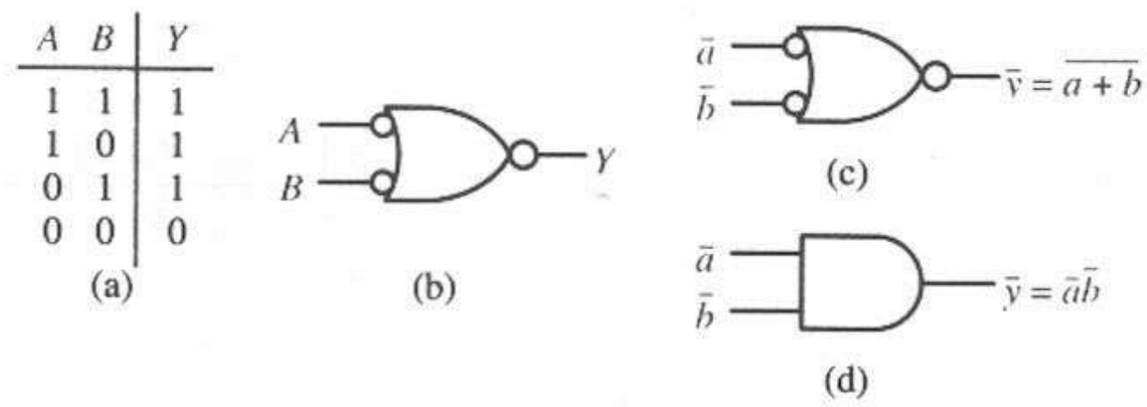


Figura 2.10 Uso de la compuerta AND en un sistema con lógica negativa. **(a)** Tabla de verdad de la compuerta AND ($L = 1$, $H = 0$). **(b)** Símbolo alternativo de la compuerta AND (lógica negativa). **(c)** Uso preferido. **(d)** Uso inadecuado.

realizada por una compuerta AND en un sistema con lógica negativa mejor que el símbolo estándar de la figura 2.6c.

Por ejemplo, consideremos la compuerta de la figura 2.10c. Las entradas de la compuerta están conectadas a señales bajas activas $\bar{a}$ y $\bar{b}$ y la salida a la señal baja activa $\bar{y}$. Formamos la expresión lógica para la salida $\bar{y}$ tomando el complemento de cada una de las entradas, aplicando un OR y complementando el resultado, como sigue:

$$\begin{aligned} \bar{y} &= \overline{\overline{(\bar{a})} + \overline{(\bar{b})}} \\ &= \overline{a + b} \\ &= \bar{f}_{OR}(a, b) \end{aligned} \tag{2.15}$$

Por tanto, $\bar{y}$ está afirmada (baja) cuando una o ambas entradas están afirmadas. Observe que la forma alternativa, que aparece en la figura 2.10d, no es incorrecta pero es más difícil de analizar y, por tanto, debemos evitarla al utilizar lógica negativa.

De manera similar, una compuerta OR realiza el operador lógico AND cuando sus entradas y salida son señales bajas activas. Podemos obtener la función realizada por una compuerta OR en un sistema con lógica negativa al sustituir 0 por H y 1 por L en la tabla de verdad de la compuerta OR de la figura 2.7b. La tabla resultante, que aparece en la figura 2.11a, es idéntica a la tabla de verdad del operador AND de la figura 2.6a. Por tanto, podemos pensar que una compuerta OR con entradas y salida bajas activas realiza la función lógica AND. Podemos verificar esto mediante el álgebra de conmutación, como hicimos antes para la compuerta AND.

$$\begin{aligned} y &= a + b \\ &= \overline{\overline{a + b}} \\ &= \overline{\bar{a} \cdot \bar{b}} \\ &= \bar{f}_{AND}(\bar{a}, \bar{b}) \end{aligned} \tag{2.16}$$

La ecuación 2.16 indica que podemos dibujar un símbolo de compuerta OR como una función AND con entradas y salida bajas activas, como se muestra en la figura 2.11b. Este símbolo alternativo ilustra la función realizada por una compuerta OR en un sistema con lógica negativa mejor que el símbolo estándar de la figura 2.7c.

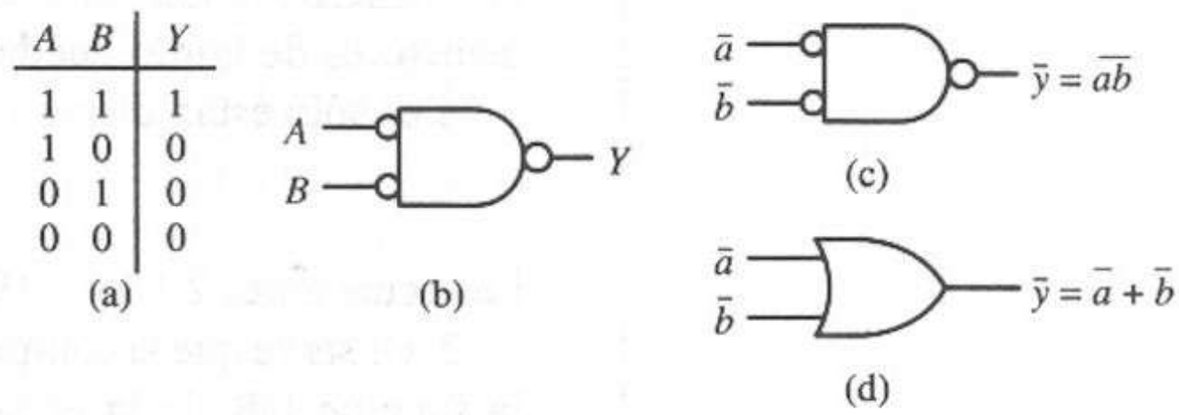


Figura 2.11 Uso de la compuerta OR en un sistema con lógica negativa. **(a)** Tabla de verdad de la compuerta OR ($L = 1$, $H = 0$). **(b)** Símbolo alternativo de la compuerta OR (lógica negativa). **(c)** Uso preferido. **(d)** Uso inadecuado.

Por ejemplo, consideremos la compuerta de la figura 2.11c. Si escribimos la expresión lógica para la salida baja activa $\bar{y}$ en términos de las entradas bajas activas $\bar{a}$ y $\bar{b}$ obtenemos

$$\begin{aligned} \bar{y} &= \overline{\overline{(\bar{a})} \cdot \overline{(\bar{b})}} \\ &= \overline{a \cdot b} \\ &= \bar{f}_{AND}(a, b) \end{aligned} \tag{2.17}$$

Por tanto, afirmamos $\bar{y}$ sólo cuando $\bar{a}$ y $\bar{b}$ se afirman en forma simultánea. Esta operación es más difícil de ver si la compuerta se dibuja como en la figura 2.11d. En consecuencia, siempre debemos utilizar la forma de la figura 2.11c con la lógica negativa.

El siguiente ejemplo demuestra el uso de las compuertas AND y OR con los dispositivos que tienen entradas y salidas bajas activas.

EJEMPLO 2.32

Diseñar un circuito lógico que implante un sistema de alarma contra incendios en un edificio.

El edificio debe estar protegido por un sistema de alarma contra incendios que tenga dos detectores de humo, un aspersor y un marcador telefónico automático que llame a los bomberos. El aspersor se debe activar si cualquiera de los detectores detecta humo, y hay que llamar a los bomberos si ambos detectores detectan humo. Los detectores tienen salidas bajas activas, $\overline{D1}$ y $\overline{D2}$, que se afirman siempre que detecten partículas de humo. El aspersor tiene una entrada baja activa $\overline{SPK}$ que debe afirmarse para que el aspersor se active. De manera similar, el marcador telefónico inicia una llamada cuando su señal de entrada baja activa $\overline{DIAL}$ se afirma.

Obtenemos las ecuaciones lógicas para el aspersor y el marcador telefónico determinando las condiciones que deben activar cada dispositivo. El aspersor debe activarse cuando cualquiera de las salidas de los detectores de humo quede afirmada. La operación deseada es $SPK = D1 + D2$. Puesto que estas señales sólo están disponibles en forma baja activa, escribimos

$$\overline{SPK} = \overline{D1 + D2} \tag{2.18}$$

De manera similar, el marcador debe activarse cuando las salidas de los dos detectores de humo quedan afirmadas; así, $DIAL = D1 \cdot D2$. Puesto que estas señales sólo están disponibles en forma baja activa, escribimos

Las ecuaciones 2.18 y 2.19 se realizan mediante el diagrama lógico de la figura 2.12. Observe que la compuerta $G1$ es una compuerta AND utilizada para realizar la función OR de la ecuación 2.18, mientras que $G2$ es una compuerta OR utilizada para realizar la función AND de la ecuación 2.19.

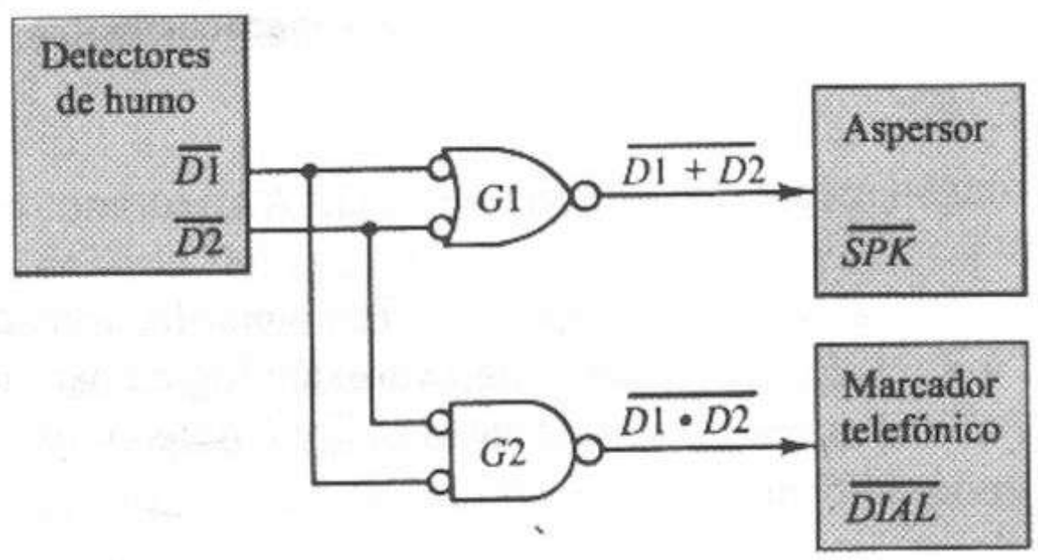


Figura 2.12 Sistema de alarma contra incendios, que ilustra la lógica negativa.

Las compuertas AND y OR se utilizan siempre que las entradas y salidas tienen la misma polaridad. Las dos compuertas que presentaremos enseguida, NAND y NOR, se utilizan en los sistemas con lógica mixta, es decir, cuando las entradas son altas activas y las salidas bajas activas, o viceversa.

NAND

La compuerta NAND es una combinación de una compuerta AND seguida de una compuerta NOT. Definimos la función NAND como

$$f_{NAND}(a, b) = \overline{ab} \tag{2.20}$$

De esta ecuación, queda claro que la compuerta NAND realiza la función lógica AND cuando sus señales de entrada son altas activas y su salida baja activa. Obtenemos las tablas de verdad para la función NAND y la compuerta NAND complementando las columnas de salida de las tablas de verdad para la función y la compuerta AND, respectivamente. Las tablas resultantes aparecen en las figuras 2.13a y b. La clave para entender la función NAND es observar que la salida es 0 si y sólo si sus entradas son simultáneamente 1.

Las figuras 2.13c, d y e muestran los símbolos estándar para la compuerta NAND. La burbuja en la terminal de salida de la figura 2.13c indica la operación NOT, estableciendo la diferencia respecto a la compuerta AND. Obtenemos la forma de la figura 2.13d aplicando el teorema de DeMorgan a la expresión de conmutación de la función NAND de la ecuación 2.20:

$$f_{\text{NAND}}(a, b) = \overline{ab} = \bar{a} + \bar{b} \tag{2.21}$$

a	b	$f_{\text{NAND}}(a, b) = \overline{ab}$
0	0	1
0	1	1
1	0	1
1	1	0

(a)

A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

(b)

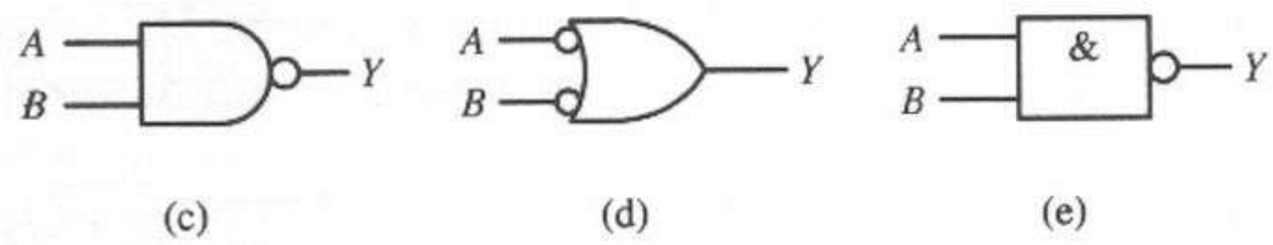


Figura 2.13 La función lógica NAND y la compuerta NAND. **(a)** La función lógica NAND. **(b)** Compuerta NAND electrónica. **(c)** Símbolo estándar. **(d)** Símbolo alternativo. **(e)** Símbolo de bloque IEEE.

Así, utilizamos una compuerta NAND para realizar la función OR cuando las señales de entrada son bajas activas y la salida es alta activa. Como explicamos en el caso de la compuerta NOT, las burbujas en el símbolo de la compuerta NAND siempre deben coincidir con las señales bajas activas. Así, utilizamos el símbolo de la figura 2.13c cuando la señal de salida es baja activa, y el de la figura 2.13d cuando las señales de entrada son bajas activas. Las figuras 2.14a y b muestran el uso correcto e incorrecto, respectivamente, de los dos símbolos de compuerta NAND.

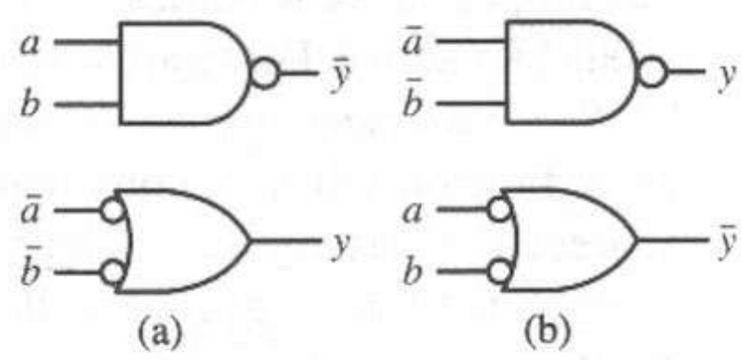


Figura 2.14 Concordancia de la polaridad de la señal con las entradas y salidas de la compuerta NAND. **(a)** Uso preferido. **(b)** Uso inadecuado.

A continuación mostramos otras propiedades interesantes de la compuerta NAND:

$$f_{\text{NAND}}(a, a) = \overline{a \cdot a} = \bar{a} \qquad = f_{\text{NOT}}(a)$$
$$f_{\text{NAND}}(a, b) = \overline{\overline{a \cdot b}} = a \cdot b \quad = f_{\text{AND}}(a, b)$$
$$f_{\text{NAND}}(\bar{a}, \bar{b}) = \overline{\bar{a} \cdot \bar{b}} = a + b = f_{\text{OR}}(a, b)$$

Por tanto, una compuerta NAND con ambas entradas controladas por la misma señal equivale a una compuerta NOT; una compuerta NAND cuya salida se complementa equivale a una compuerta AND, y una compuerta NAND con entradas complementadas actúa como una compuerta OR.

Así, podemos utilizar las compuertas NAND para implantar los tres operadores elementales (AND, OR y NOT), como se muestra en la figura 2.15. En consecuencia, podemos construir cualquier función de conmutación, utilizando sólo compuertas NAND. Las compuertas con esta propiedad se llaman *primitivas* o *funcionalmente completas*.

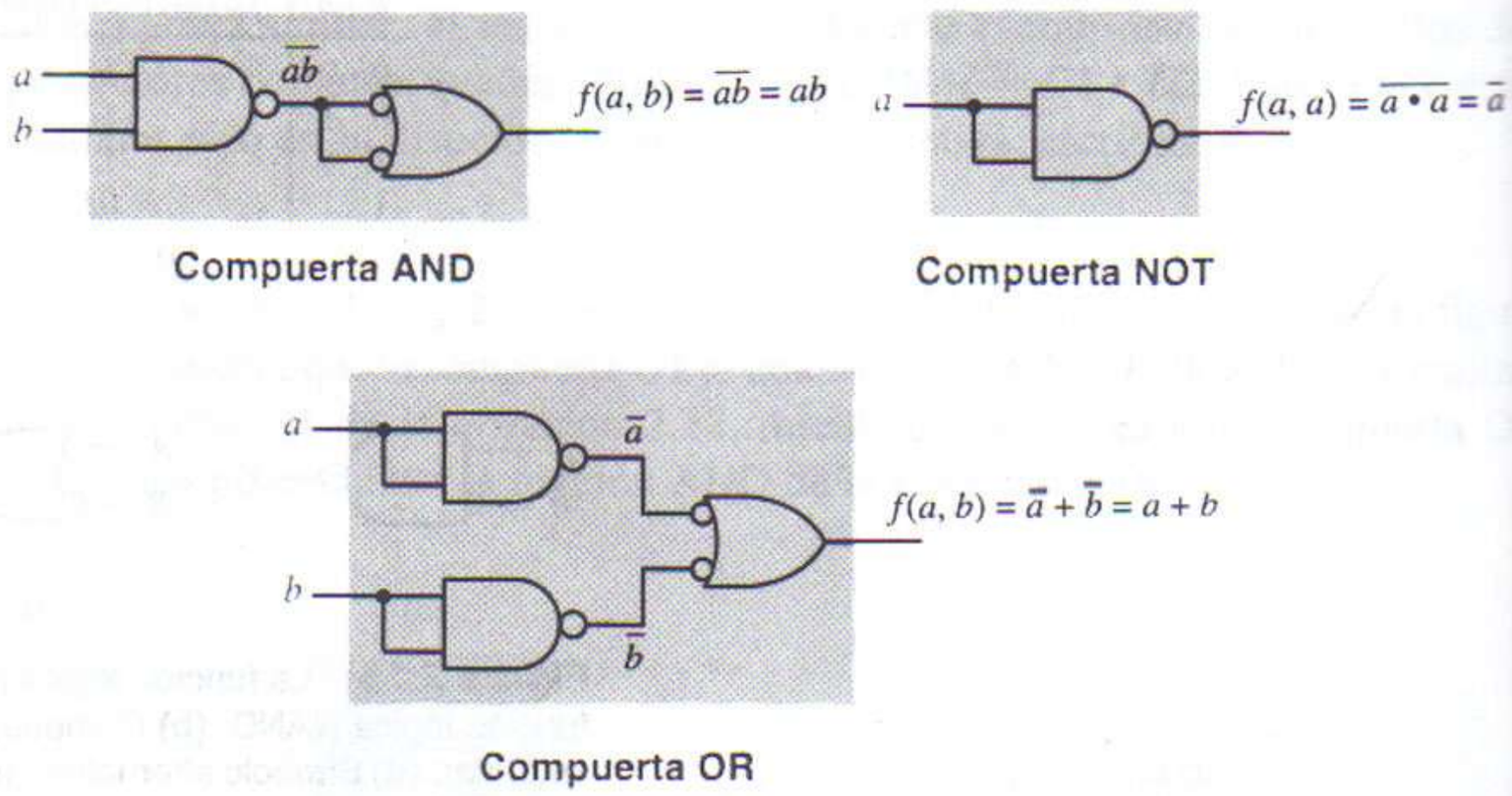


Figura 2.15 Compuertas AND, OR y NOT construidas exclusivamente mediante compuertas NAND.

NOR

La compuerta NOR es una combinación de compuerta OR seguida de una compuerta NOT, lo que representa la función

$$f_{\text{NOR}}(a, b) = \overline{a + b} \tag{2.22}$$

La compuerta NOR realiza la función lógica OR con entradas altas activas y una salida baja activa. Por tanto, la tabla de verdad para la función NOR y la compuerta NOR se obtienen complementando las columnas de salida de las tablas de verdad de la función OR y la compuerta OR, respectivamente. Las tablas resultantes aparecen en las figuras 2.16a y b. La clave para recordar la función de una compuerta NOR es la primera fila de la tabla de verdad; la salida de una compuerta NOR es 1 si y sólo si ambas entradas son simultáneamente 0.

a	b	$f_{\text{NOR}}(a, b) = \overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0

(a)

A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

(b)

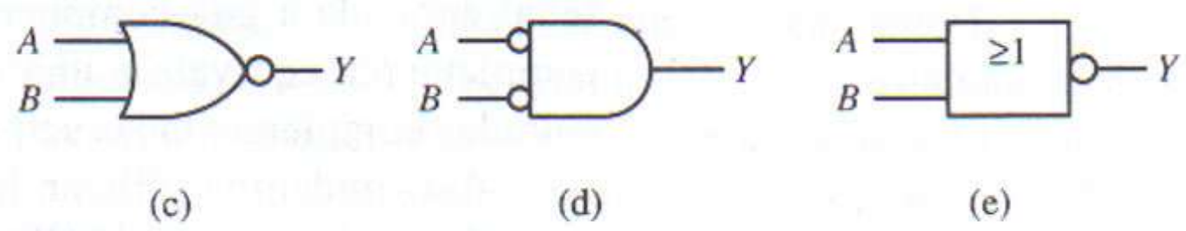


Figura 2.16 La función lógica NOR y la compuerta NOR. **(a)** La función lógica NOR. **(b)** Compuerta NOR electrónica. **(c)** Símbolo estándar. **(d)** Símbolo alternativo. **(e)** Símbolo de bloque IEEE.

Los símbolos estándar para la compuerta NOR se presentan en las figuras 2.16c, d y e. La burbuja en la terminal de salida indica la operación NOT, lo que establece su diferencia con la compuerta OR. Obtenemos la forma de la figura 2.16d aplicando el teorema de DeMorgan a la definición de la función NOR definida mediante la ecuación 2.22:

$$f_{\text{NOR}}(a, b) = \overline{a + b} = \bar{a} \cdot \bar{b} \tag{2.23}$$

Así, podemos utilizar una compuerta NOR para realizar la función AND con entradas bajas activas y una salida alta activa. Como en el caso de la compuerta NAND, utilizamos el símbolo de la figura 2.16c cuando la señal de salida es baja activa, y el de la figura 2.16d, cuando las señales de entrada son bajas activas. Las figuras 2.17a y b ilustran el uso correcto e incorrecto de los símbolos para la compuerta NOR, respectivamente.

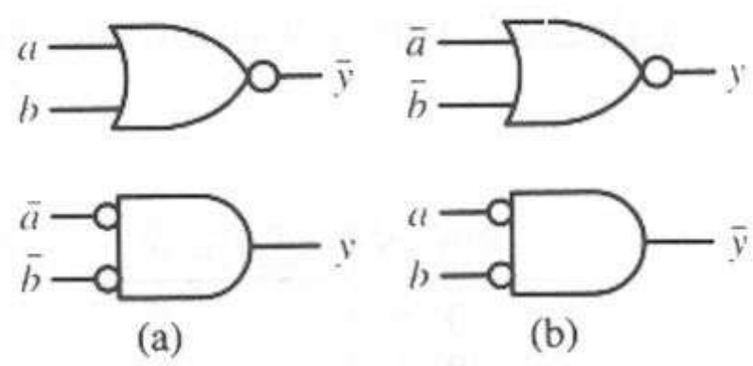


Figura 2.17 Concordancia de la polaridad de la señal con las entradas y salidas de la compuerta NOR. **(a)** Uso preferido. **(b)** Uso inadecuado.

Como en el caso de las compuertas NAND, las compuertas NOR también son elementos primitivos, en el sentido de que pueden servir para generar las operaciones AND, OR y NOT, como mostramos a continuación.

$$\begin{aligned} f_{\text{NOR}}(a, a) &= \overline{a + a} = \bar{a} &&= f_{\text{NOT}}(a) \\ \bar{f}_{\text{NOR}}(a, b) &= \overline{\overline{a + b}} = a + b &&= f_{\text{OR}}(a, b) \\ f_{\text{NOR}}(\bar{a}, \bar{b}) &= \overline{\bar{a} + \bar{b}} = a \cdot b &&= f_{\text{AND}}(a, b) \end{aligned}$$

La figura 2.18 presenta estas tres operaciones en forma simbólica.

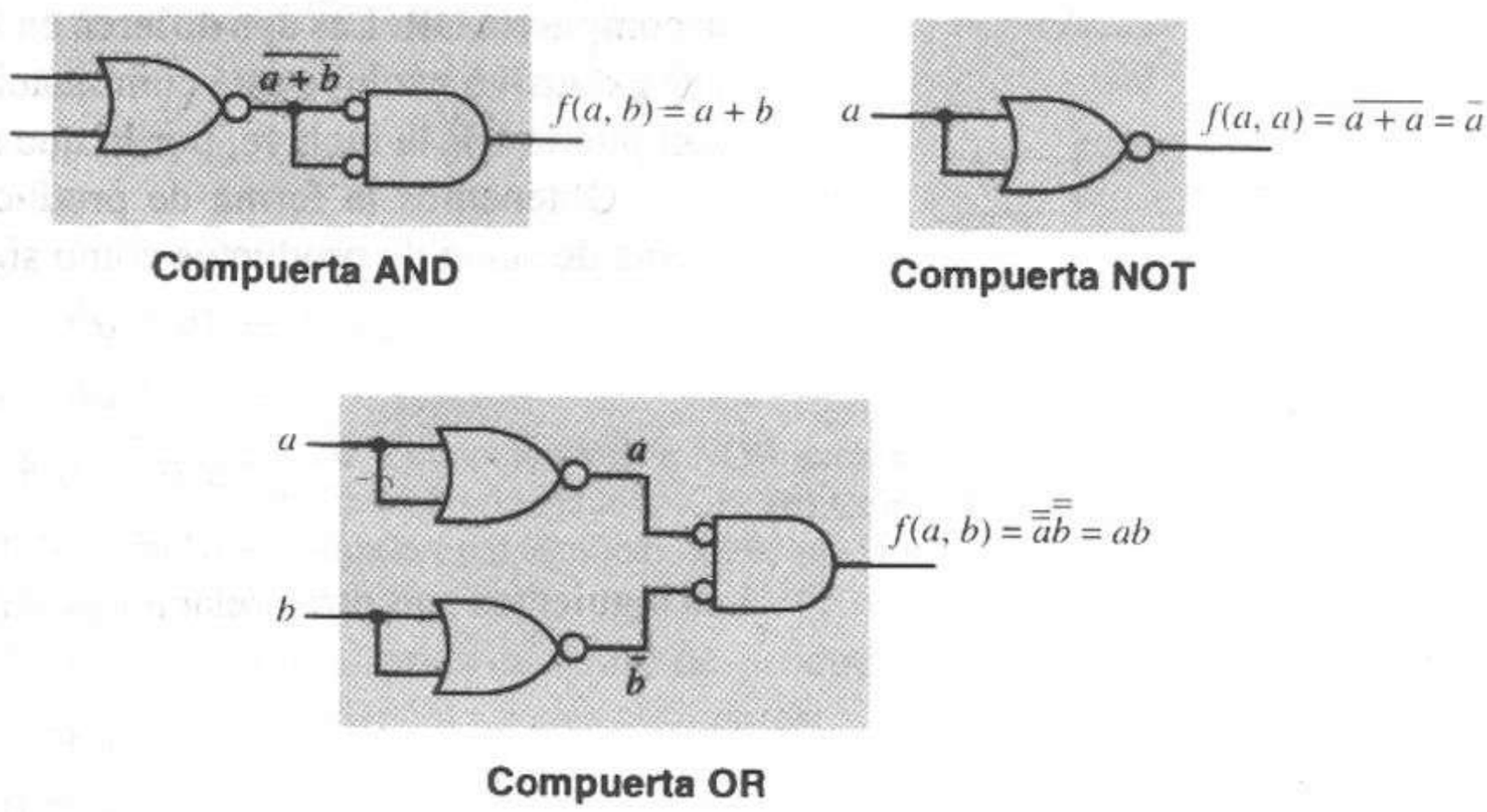


Figura 2.18 Compuertas AND, OR y NOT construidas exclusivamente mediante compuertas NOR.

Como ambas compuertas, NAND y NOR, son funcionalmente completas, resultan valiosas porque se puede implantar un diseño completo usando un único tipo de elemento. Es más fácil construir un circuito integrado si todas sus compuertas son NAND (o todas NOR) en vez de combinar compuertas AND, OR y NOT. Además, los circuitos con compuertas electrónicas NAND y NOR son por lo general más rápidos y fáciles de fabricar que las compuertas AND y OR equivalentes y, por tanto, tienen un efecto importante sobre el costo.

OR exclusivo (XOR)

La operación de OR exclusivo (o XOR) se define de manera funcional como

$$f_{XOR}(a, b) = a \oplus b = a\bar{b} + \bar{a}b \tag{2.24}$$

La tabla de verdad que se obtiene de la ecuación 2.24 aparece en la figura 2.19a. La tabla de verdad correspondiente para la compuerta XOR aparece en la figura 2.19b, y los símbolos lógicos estándar, en las figuras 2.19c y d.

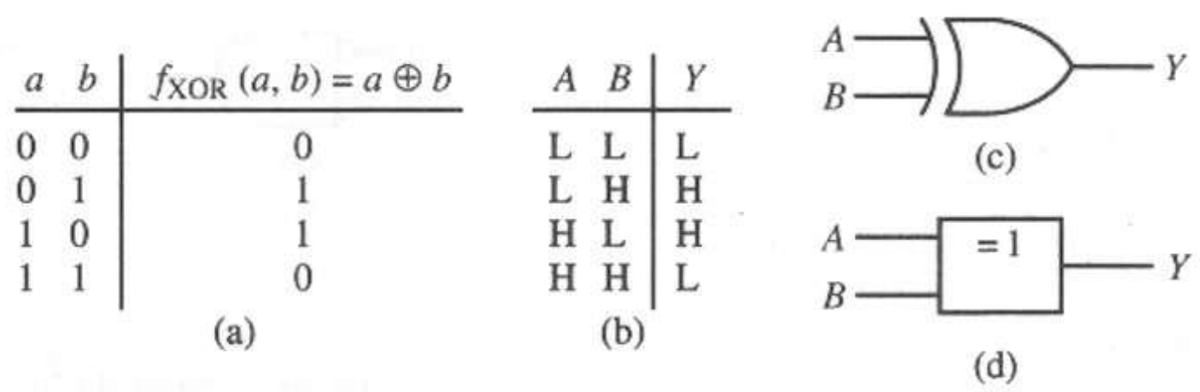


Figura 2.19 La función lógica OR exclusivo (XOR) y la compuerta XOR. **(a)** La función lógica XOR. **(b)** Compuerta XOR electrónica. **(c)** Símbolo estándar. **(d)** Símbolo de bloque IEEE.

La salida de la compuerta OR exclusivo es 1 si y sólo si sus entradas no son iguales en forma simultánea. En otras palabras, cuando las entradas son diferentes, la salida es 1. El OR exclusivo se llama así debido a su relación con la compuerta OR. Las dos difieren en la combinación de entradas $a = 1, b = 1$. El OR exclusivo excluye esta combinación, dando como salida 0, mientras que la compuerta OR la incluye, por lo que se le llama también OR-inclusivo.

Obtenemos la forma de producto de sumas para el OR-exclusivo de la forma de suma de productos como sigue:

$$\begin{aligned} a \oplus b &= \bar{a}b + a\bar{b} \\ &= \bar{a}a + \bar{a}b + a\bar{b} + b\bar{b} && \text{[P2(a), P6(b)]} \\ &= \bar{a}(a + b) + \bar{b}(a + b) && \text{[P5(b)]} \\ &= (\bar{a} + \bar{b})(a + b) && \text{[P5(b)]} \end{aligned}$$

Las siguientes son otras relaciones útiles relativas al OR-exclusivo:

$$a \oplus a = 0 \tag{2.25}$$

$$a \oplus \bar{a} = 1 \tag{2.26}$$

$$a \oplus 0 = a \tag{2.27}$$

$$a \oplus 1 = \bar{a} \tag{2.28}$$

$$\bar{a} \oplus \bar{b} = a \oplus b \tag{2.29}$$

$$a \oplus b = b \oplus a \tag{2.30}$$

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c \tag{2.31}$$

El lector puede verificar la validez de estas relaciones construyendo sus tablas de verdad.

El símbolo de bloque estándar IEEE para la compuerta de OR-exclusivo indica que la salida se afirma cuando la suma matemática de las entradas es igual a 1:

$a\ b$	$\text{sum}(a, b)$	$\text{sum}(a, b) = 1?$	$f(a, b) = a \oplus b$
0 0	0	Falso	0
0 1	1	Verdadero	1
1 0	1	Verdadero	1
1 1	2	Falso	0

Podemos ver de esta tabla que la salida de una compuerta OR-exclusivo es la suma módulo 2 de sus entradas. Por esta razón, las compuertas OR-exclusivo se utilizan con frecuencia en el diseño de circuitos aritméticos que realizan sumas y restas binarias. En el capítulo 4 analizaremos esto con mayor detalle.

NOR-exclusivo (XNOR)

Una función común relacionada con el OR-exclusivo es la operación de coincidencia, o NOR-exclusivo (XNOR), que es sencillamente el complemento del OR-exclusivo. Definimos esta función como:

$$f_{\text{XNOR}}(a, b) = \overline{a \oplus b} = a \odot b \tag{2.32}$$

Las tablas de verdad y los símbolos lógicos de la compuerta XNOR aparecen en

Mantenimos XOR
$(a+b)(\bar{a}+\bar{b})$

a	b	$f_{\text{XNOR}}(a, b) = a \odot b$
0	0	1
0	1	0
1	0	0
1	1	1

(a)

A	B	Y
L	L	H
L	H	L
H	L	L
H	H	H

(b)

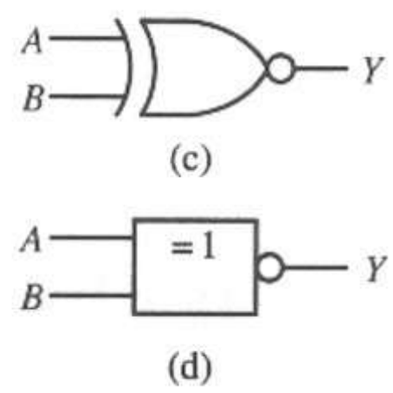


Figura 2.20 La función lógica NOR exclusivo (XNOR) y la compuerta XNOR. **(a)** La función lógica XNOR. **(b)** Compuerta XNOR electrónica. **(c)** Símbolo estándar. **(d)** Símbolo de bloque IEEE.

la figura 2.20. Las formas de suma de productos y producto de sumas de la operación de coincidencia son las siguientes:

$$\begin{aligned} a \odot b &= \overline{a \oplus b} \\ &= \overline{\bar{a}b + a\bar{b}} && \text{[P2]} \\ &= \overline{\bar{a}b} \cdot \overline{a\bar{b}} && \text{[T8(a)]} \end{aligned}$$

$$= (a + \bar{b})(\bar{a} + b) \qquad [T8(b)]$$
$$= a\bar{a} + ab + \bar{a}\bar{b} + \bar{b}b \qquad [P5(b)]$$
$$= ab + \bar{a}\bar{b} \qquad [P6(b), P2(a)]$$

También se puede verificar fácilmente que

$$a \oplus \bar{b} = a \odot b \tag{2.33}$$

2.4 Análisis de circuitos combinatorios

Los circuitos digitales se diseñan transformando una descripción verbal de una función en un conjunto de ecuaciones de conmutación y realizando después las ecuaciones mediante compuertas, dispositivos lógicos programables (PLD) u otros elementos lógicos. El análisis de los circuitos digitales es el problema inverso. Partimos de una realización en hardware de un circuito digital y obtenemos una descripción del circuito en forma de expresiones de conmutación, tablas de verdad, diagramas de tiempos, u otras descripciones de su comportamiento. Utilizamos el análisis para determinar el comportamiento de un circuito lógico, para verificar que el comportamiento de un circuito corresponda con sus especificaciones, o como apoyo para convertir el circuito a una forma diferente, ya sea con el fin de reducir el número de compuertas o para realizarlo con diferentes elementos.

En este capítulo presentamos el análisis y síntesis de los circuitos digitales, incluidos el diseño y uso de módulos de bloques de construcción que sirven para implantar diseños de mayor tamaño. En el capítulo 4 analizaremos módulos más complejos, con integración a mediana escala (MSI). Estos módulos son dispositivos de mayor nivel que contienen de 100 a 1000 transistores. En el capítulo 5 examinaremos el uso de los dispositivos lógicos programables para diseñar circuitos digitales.

2.4.1 Método algebraico

Podemos construir redes lógicas interconectando las compuertas presentadas en la sección anterior. Utilizamos estos circuitos para realizar funciones específicas dentro de un sistema digital de cómputo. Cualquier red de conmutación se puede representar por completo mediante una expresión o función de conmutación, y con ello podemos aplicar toda la capacidad del álgebra de conmutación para llevar la función de conmutación a una forma deseada.

Hay que recordar que todas las expresiones de conmutación se pueden escribir en términos de las operaciones AND, OR y NOT. Por tanto, podemos construir cualquier red de conmutación utilizando sólo elementos primitivos, como las compuertas NAND (o las NOR), como se muestra en la figura 2.15 (o la figura 2.18).

EMPLO 2.33

Determinar una expresión de conmutación simplificada y la red lógica para el circuito lógico de la figura 2.21a.

Escribimos una expresión de conmutación para la salida de cada compuerta.

$$P_1 = \overline{ab}$$
$$P_2 = \overline{\bar{a} + c}$$

$$P_3 = b \oplus \bar{c}$$
$$P_4 = P_1 \cdot P_2 = \overline{ab}\,\overline{(\bar{a} + c)}$$

La salida es

$$f(a, b, c) = \overline{P_3 + P_4}$$
$$= \overline{(b \oplus \bar{c}) + \overline{ab}\,\overline{(\bar{a} + c)}}$$

Para analizar esta función, podemos llevarla a una forma más sencilla empleando el álgebra de conmutación:

$$\begin{aligned}
\bar{f}(a, b, c) &= (b \oplus \bar{c}) + \overline{ab}\,\overline{\bar{a} + c} \\
&= bc + \bar{b}\bar{c} + \overline{ab}\,\overline{\bar{a} + c} && \text{[Ec. 2.24]} \\
&= bc + \bar{b}\bar{c} + (\bar{a} + \bar{b})a\bar{c} && \text{[T8]} \\
&= bc + \bar{b}\bar{c} + a\bar{b}\bar{c} && \text{[T5(b)]} \\
&= bc + \bar{b}\bar{c} && \text{[T4(a)]} \\
\bar{f}(a, b, c) &= b \odot c && \text{[Ec. 2.32]}
\end{aligned}$$

Por tanto, de la ecuación 2.32,

$$f(a, b, c) = \overline{b \odot c} = b \oplus c$$

Hemos reducido esta función a una compuerta OR exclusivo, que se muestra en la figura 2.21b. Las dos redes de conmutación de la figura 2.21 tienen la misma

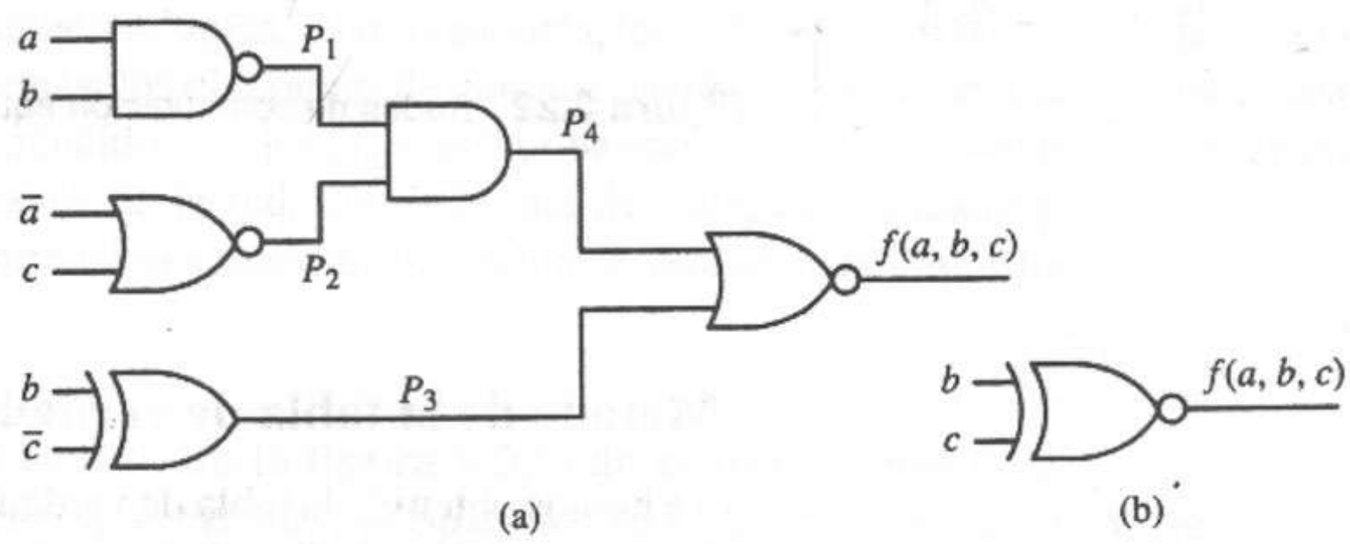


Figura 2.21 Redes de conmutación equivalentes.

tabla de verdad y, por ende, son equivalentes. Queda claro que la figura 2.21b es más recomendable, pues es menos compleja.

EJEMPLO 2.34

Determinar una expresión de conmutación y un circuito simplificados para la red de la figura 2.22.

La expresión lógica para cada salida de la compuerta aparece en el diagrama lógico de la figura 2.22. A partir de ellas, podemos obtener la expresión de salida como

$$\begin{aligned}
&f(a; b, c) \\
&\quad = \overline{\overline{(a \oplus b)(b \oplus c)} \cdot (\overline{\bar{a} + \bar{b}} + \overline{a + c})} \\
&\quad = \overline{\overline{(a \oplus b)(b \oplus c)}} + \overline{\overline{\bar{a} + \bar{b}} + \overline{a + c}} && \text{[T8(b)]}
\end{aligned}$$

$$
\begin{aligned}
&= (a \oplus b)(b \oplus c) + (\bar{a} + \bar{b})(a + c) && \text{[T8(a)]}\\
&= (a\bar{b} + \bar{a}b)(b\bar{c} + \bar{b}c) + (\bar{a} + \bar{b})(a + c) && \text{[Ec. 2.24]}\\
&= a\bar{b}b\bar{c} + a\bar{b}\bar{b}c + \bar{a}bb\bar{c} + \bar{a}b\bar{b}c + \bar{a}a + \bar{a}c + a\bar{b} + \bar{b}c && \text{[P5(b)]}\\
&= a\bar{b}c + \bar{a}b\bar{c} + \bar{a}c + a\bar{b} + \bar{b}c && \text{[P6(b),T4(a)]}\\
&= \bar{a}b\bar{c} + \bar{a}c + a\bar{b} + \bar{b}c && \text{[T4(a)]}\\
&= \bar{a}b\bar{c} + \bar{a}c + a\bar{b} && \text{[T9(a)]}\\
&= \bar{a}b + \bar{a}c + a\bar{b} && \text{[T7(a)]}\\
&= \bar{a}c + a \oplus b && \text{[Ec. 2.24]}
\end{aligned}
$$

Observe que ésta es la forma de la red de conmutación de la figura 2.22b.

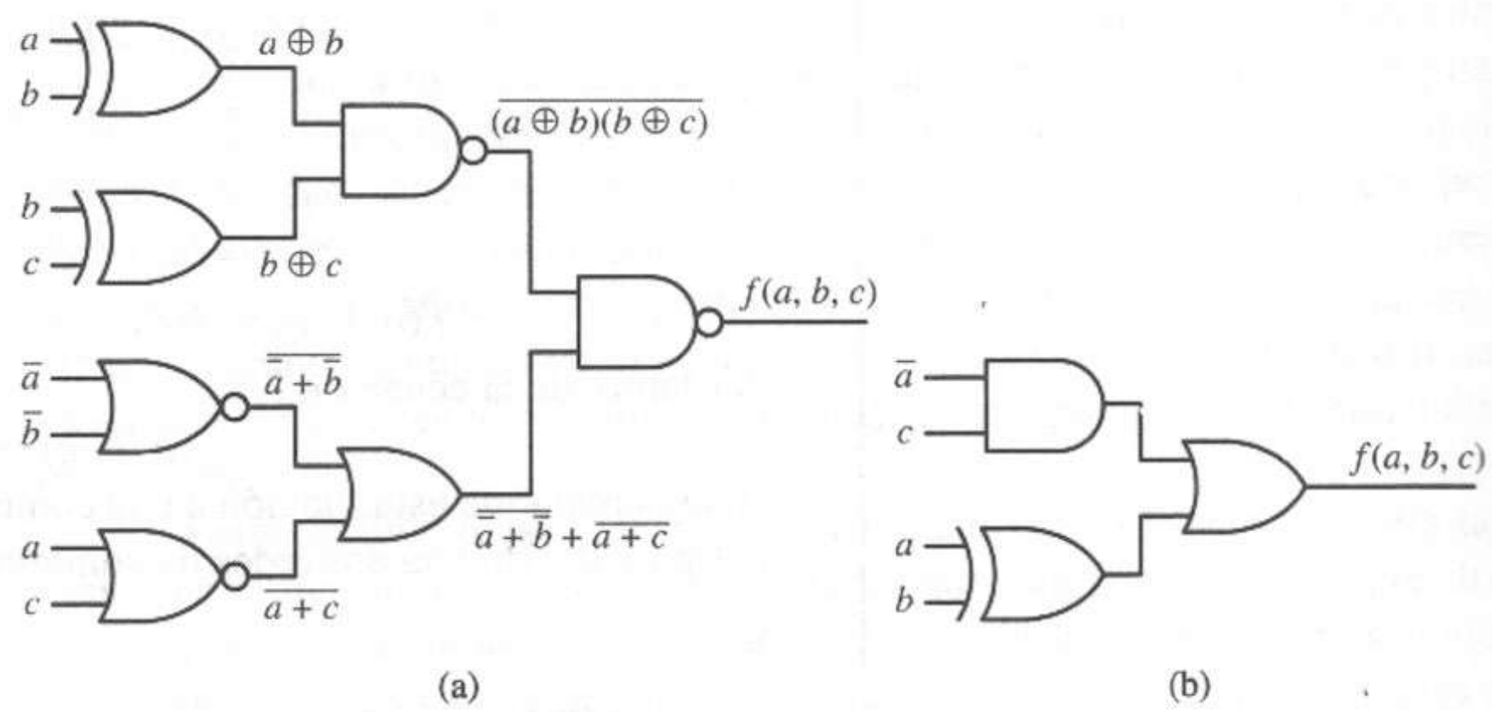


Figura 2.22 Redes de conmutación equivalentes.

Método de la tabla de verdad

Ya hemos obtenido la tabla de verdad para una función a partir de una expresión de conmutación evaluando la expresión, parte por parte. Es posible seguir el mismo método con los diagramas lógicos, obteniendo la tabla de verdad compuerta por compuerta.

Podemos obtener la tabla de verdad para la función del último ejemplo utilizando la técnica del álgebra de conmutación ya descrita:

abc	$\bar{a}c$	$a \oplus b$	$f(a, b, c)$
0 0 0	0	0	0
0 0 1	1	0	1
0 1 0	0	1	1
0 1 1	1	1	1
1 0 0	0	1	1
1 0 1	0	1	1
1 1 0	0	0	0
1 1 1	0	0	0

La columna $\bar{a}c$ es 1 siempre que $\bar{a} = 1$ y $c = 1$, o siempre que $a = 0$ y $c = 1$. La columna $a \oplus b$ es 1 siempre que $a \neq b$. Obtenemos el OR de estas dos columnas para crear $f(a, b, c)$; por tanto, $f(a, b, c)$ vale 0 si $\bar{a}c$ y $a \oplus b$ son ambos 0.

En esta sección consideramos una red de conmutación, la analizamos escribiendo la función de conmutación, la simplificamos mediante el álgebra de conmutación, y obtuvimos una red equivalente, pero menos compleja.

2.4.2 Análisis de diagramas de tiempos

Hasta ahora, hemos analizado los diagramas de circuitos lógicos obteniendo expresiones o tablas de verdad para las funciones lógicas realizadas por los circuitos. Otro método de análisis consiste en aplicar una serie de valores a las entradas de un circuito en cierto lapso, ya sea en forma experimental o con un programa de simulación lógica, y tomar nota de la relación entre las entradas y la serie correspondiente de salidas en forma de un diagrama de tiempos. De este diagrama, podemos deducir la función lógica realizada por el circuito y estudiar los efectos de los retardos de propagación de las compuertas sobre el funcionamiento del circuito.

Diagramas de tiempos

Un *diagrama de tiempos* es una representación gráfica de las relaciones entre las señales de entrada y salida de una red de conmutación, como podrían verse en la pantalla de un osciloscopio o analizador lógico o en un programa de simulación lógica. Con frecuencia, también se muestran las señales intermedias. Además, los diagramas de tiempos pueden mostrar los retardos por propagación introducidos por los dispositivos de conmutación cuando las señales se propagan a través de la red. Un diagrama de tiempos adecuado puede mostrar toda la información contenida en la tabla de verdad, como muestra el ejemplo siguiente.

EJEMPLO 2.35

El circuito de la figura 2.23a se estimula con una serie de entradas, produciendo el diagrama de tiempos de la figura 2.23b. En este ejemplo, 1 se representa con una señal alta y 0 con una baja. Determinar la tabla de verdad y las listas de mintérminos para las dos funciones $f_\alpha(A, B, C)$ y $f_\beta(A, B, C)$ realizadas por este circuito.

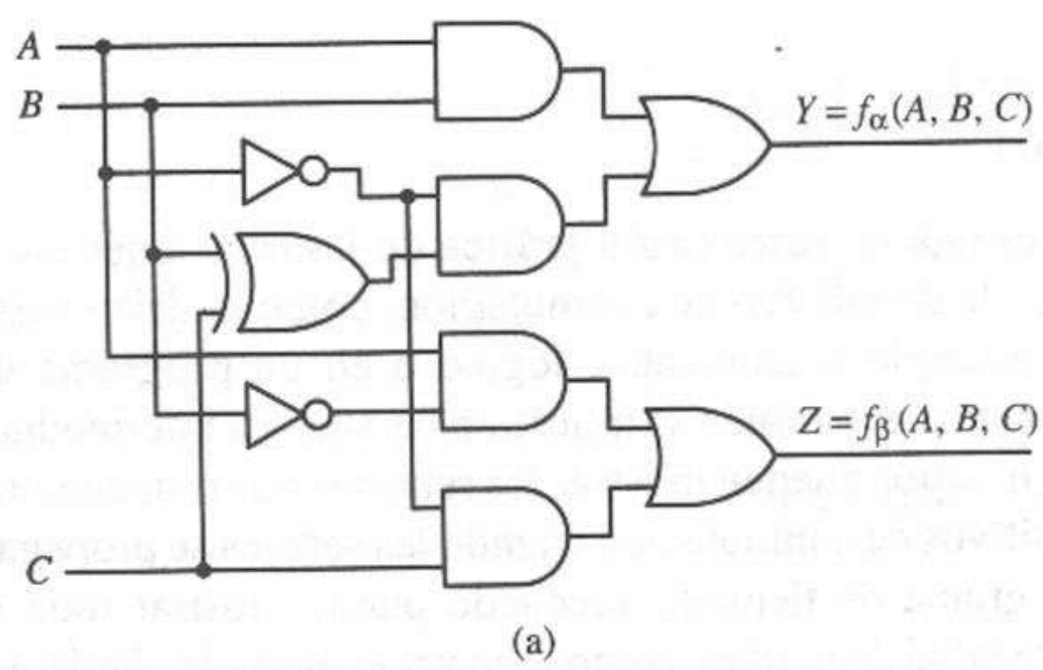
Las señales de entrada y salida se indican sobre el diagrama del circuito. Hemos seleccionado los patrones de entrada de modo que en una unidad de tiempo aparezcan todas las combinaciones posibles de las entradas A, B y C.

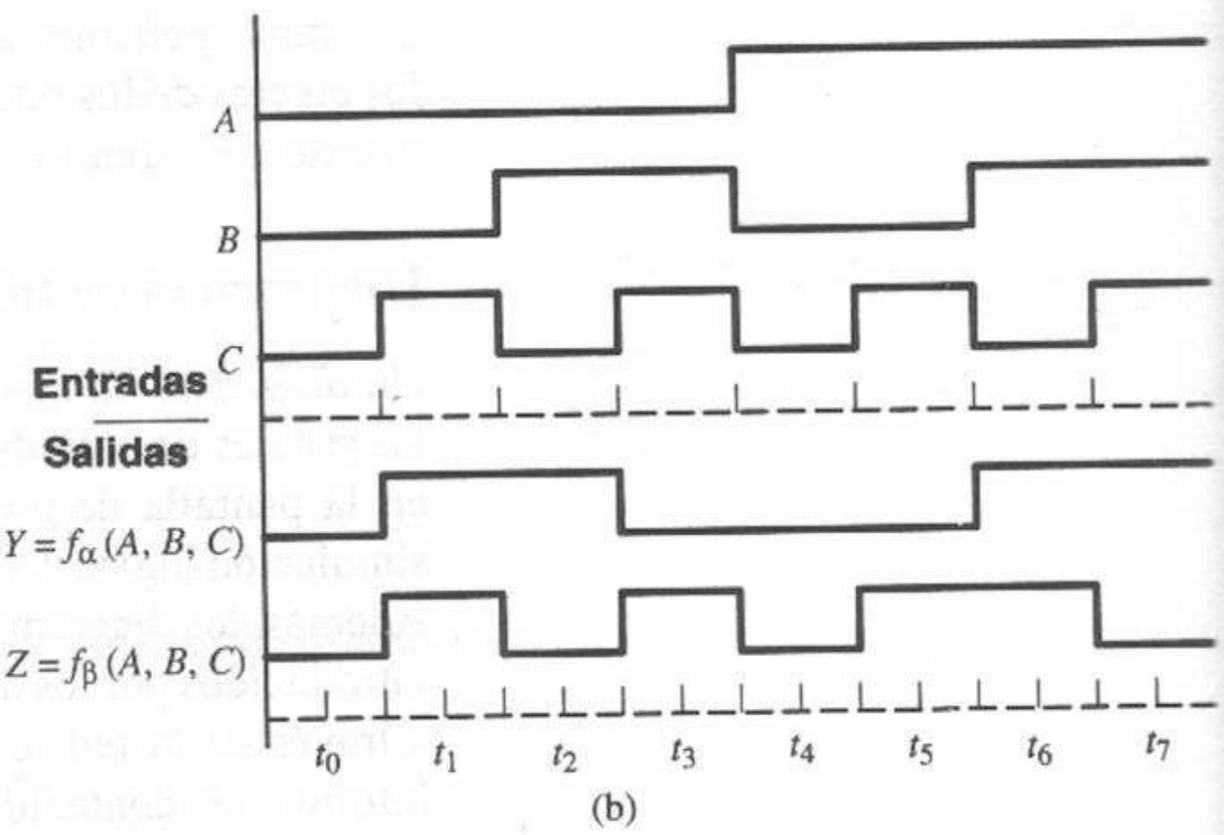
Al analizar el diagrama de tiempos en los instantes $t_0, t_1, \ldots, t_7$, determinamos los valores de las entradas y las salidas en cada instante y los escribimos en forma de tabla de verdad, como en la figura 2.23c. A partir de esta tabla de verdad, podemos escribir la lista de mintérminos y después obtener una expresión lógica simplificada para cada función, de la manera siguiente:

$$\begin{aligned} f_\alpha(A, B, C) &= \sum m(1, 2, 6, 7) \\ &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC \\ &= \bar{A}\bar{B}C + B\bar{C} + AB \end{aligned}$$

$$
\begin{aligned}
f_\beta(A, B, C) &= \sum m(1, 3, 5, 6) \\
&= \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C} \\
&= \bar{A}C + A\bar{B}C + AB\bar{C}
\end{aligned}
$$

En el ejemplo anterior, mostramos todas las salidas de las compuertas como si cambiaran en forma instantánea después de un cambio en la entrada. En realidad, siempre hay un retardo entre el momento de un cambio de entrada y el cambio correspondiente en la salida de un circuito. En la siguiente sección consideraremos los efectos de los retardos por propagación de las compuertas sobre el funcionamiento del circuito.





Instante	Entradas ABC	Salidas $f_\alpha(A, B, C)$	$f_\beta(A, B, C)$
t_0	000	0	0
t_1	001	1	1
t_2	010	1	0
t_3	011	0	1
t_4	100	0	0
t_5	101	0	1
t_6	110	1	1
t_7	111	1	0

(c)

Figura 2.23 Obtención de una tabla de verdad a partir de un diagrama de tiempos. **(a)** Circuito lógico que realiza dos funciones. **(b)** Diagrama de tiempos. **(c)** Tabla de verdad.

Retardo por propagación

Además de la función lógica, el diseñador debe preocuparse por varias características físicas de los circuitos lógicos digitales, como las siguientes:

- Retardos por propagación
- Restricciones de *fan-in* y *fan-out* de las compuertas
- Consumo de energía
- Tamaño y peso

Éstas son propiedades físicas de los circuitos de bajo nivel utilizados para crear las compuertas lógicas, y dependen del número y configuración de las compuertas en un circuito dado. El diseño de bajo nivel no es un objetivo de este libro; el lector puede consultar Wakerly [9]. Sin embargo, los retardos por propagación y las restricciones de *fan-in* y *fan-out* tienen un efecto importante sobre el diseño lógico, por lo que deben tomarse en cuenta durante cualquier análisis o diseño de cualquier circuito lógico digital.

Una compuerta lógica física requiere una cantidad no nula de tiempo para reaccionar a los cambios de entrada y producir modificaciones en su estado de salida. Este retardo entre el instante de cambio de la entrada y el cambio correspondiente en la salida es el *retardo por propagación*. Así, si un circuito lógico realiza una función $z = f(x_1,...,x_n)$, el retardo por propagación es el tiempo que los cambios tardan en "propagarse" desde alguna entrada x_i a través del circuito hasta la salida z. Los retardos por propagación dependen de la complejidad del circuito, de la tecnología electrónica utilizada, y de factores como el *fan-out* (el número de entradas de otras compuertas controladas por una única salida de la compuerta), la temperatura y el voltaje del circuito.

Después de varios cambios de entrada, las salidas de los circuitos de com puertas lógicas electrónicas pueden tardar un tiempo distinto en cambiar d bajo a alto que de alto a bajo. Por tanto, generalmente especificamos dos pará metros de retardo por propagación para una compuerta lógica dada:

t_{PLH} = tiempo de retardo por propagación, con salida de bajo a alto nivel

t_{PHL} = tiempo de retardo por propagación, con salida de alto a bajo nivel

donde t_{PLH} y t_{PHL} se miden a partir del momento de cambio en la entrada hasta el momento del cambio correspondiente en la salida.

Si no se necesita información precisa de tiempos, utilizamos un único parámetro de retardo por propagación, denotado por t_{PD}, como aproximación de t_{PLH} y t_{PHL}. Por lo general, t_{PD} se calcula como la media de t_{PLH} y t_{PHL}:

$$t_{PD} = \frac{t_{PLH} + t_{PHL}}{2}$$

Para la compuerta AND de la figura 2.24a, las figuras 2.24b-d ilustran la respuesta de la salida de la compuerta a una serie de cambios en sus valores de entrada. En la figura 2.24b se muestra el caso ideal, en el que las salidas tienen un cambio instantáneo; es decir, el retardo por propagación es nulo. En la figura 2.24c se muestran todos los cambios de salida retrasados por un retardo de propagación medio t_{PD}. La figura 2.24d presenta una imagen más precisa de los tiempos, con parámetros distintos para t_{PLH} y t_{PHL}.

La tabla 2.7 enumera los valores de t_{PD} para compuertas NAND con dos entradas de varias familias lógicas diferentes, junto con la disipación de energía por compuerta para cada una.

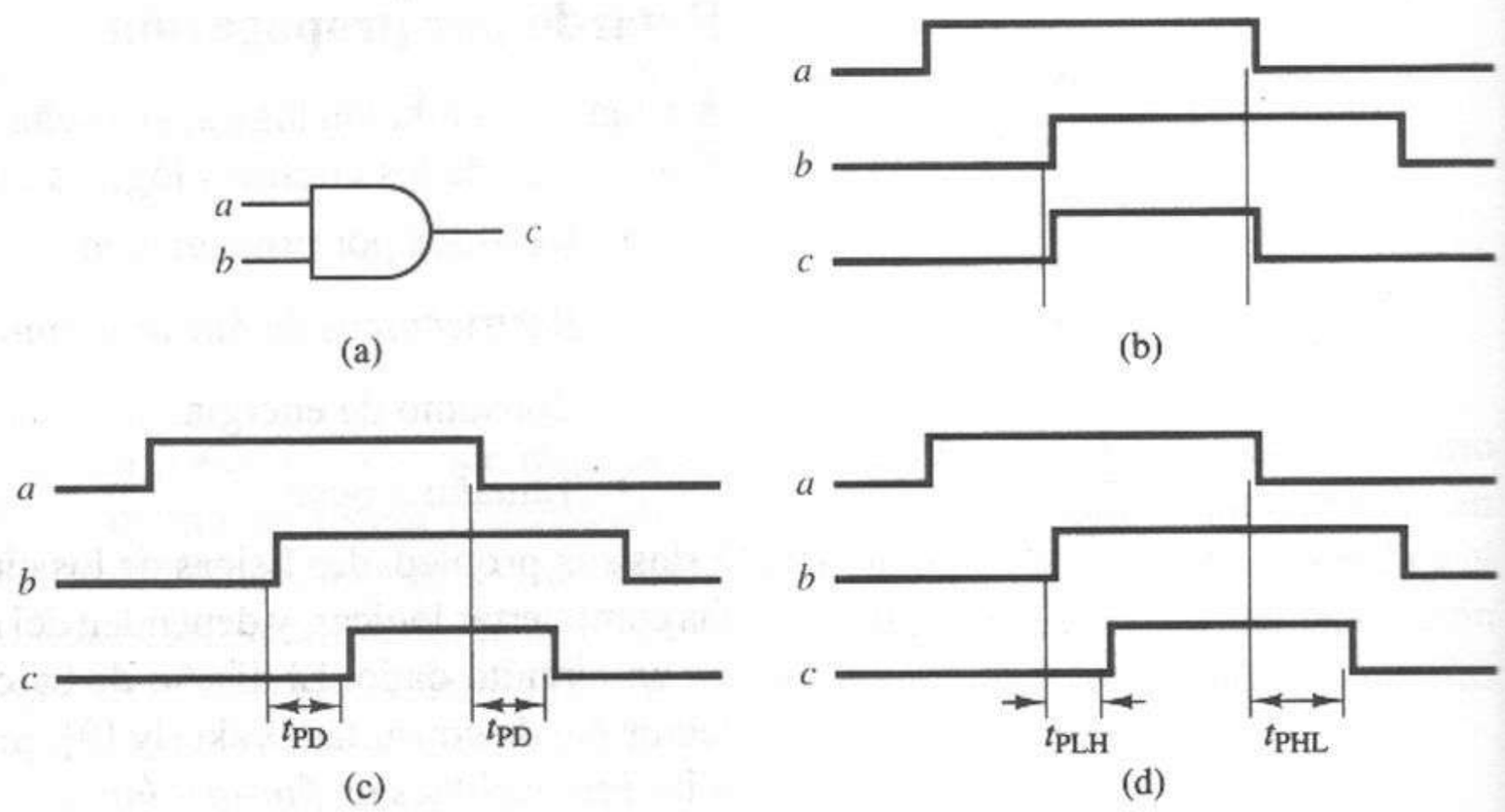


Figura 2.24 Retardo por propagación a través de una compuerta lógica. **(a)** Compuerta AND con dos entradas. **(b)** Retardo ideal (nulo). **(c)** $t_{PD} = t_{PLH} = t_{PHL}$. **(d)** $t_{PLH} < t_{PHL}$.

Se dispone de compuertas lógicas con varios tipos de tecnología, ocho de las cuales se muestran en la tabla. Como puede verse, existe cierto juego entre la velocidad y la disipación de energía en las diversas tecnologías. Por ejemplo, los dispositivos 74*S*×× son más rápidos que los dispositivos equivalentes 74×× o 74*LS*××, mientras que los dispositivos 74*LS*×× son más lentos, pero consumen menos energía. Con frecuencia, un diseñador sacrifica la velocidad en aras de un menor consumo de energía en aplicaciones en las que la corriente de la fuente de potencia es limitada, como es el caso de los sistemas energizados por baterías.

TABLA 2.7 DISIPACIÓN DE ENERGÍA Y RETARDOS POR PROPAGACIÓN PARA VARIAS FAMILIAS LÓGICAS [8]

Familia lógica	Retardo por propagación	Disipación de energía por compuerta (mW)	Tecnología
7400	10	10	TTL estándar
74H00	6	22	TTL de alta velocidad
74L00	33	1	TTL de baja potencia
74LS00	9.5	2	TTL Schottky de baja potencia
74S00	3	19	TTL Schottky
74ALS00	3.5	1.3	TTL Schottky avanzado de baja potencia
74AS00	3	8	TTL Schottky avanzado
74HC00	8	0.17	CMOS de alta velocidad

Los retardos por propagación también difieren entre las diferentes compuertas implantadas con la misma tecnología, ya que sus circuitos a nivel de transistores son diferentes. La tabla 2.8 enumera los parámetros t_{PHL} y t_{PLH} para cinco compuertas primitivas (AND, OR, NAND, NOR y NOT) de la familia

74LS. Observe que damos un valor típico y un valor máximo para cada parámetro. Los retardos por propagación varían de un dispositivo a otro, y son afectados por la cantidad de carga controlada por la compuerta. Por ello, la mayor parte de las hojas de datos de los dispositivos especifica un tiempo máximo de retardo, correspondiente a las peores condiciones de carga, además del tiempo típico de retardo para cada dispositivo.

TABLA 2.8 RETARDO POR PROPAGACIÓN DE COMPUERTAS PRIMITIVAS DE LA SERIE 74LS [8]

		t_{PLH}		t_{PHL}	
Circuito	**Función**	**Típico**	**Máximo**	**Típico**	**Máximo**
74LS04	NOT	9	15	10	15
74LS00	NAND	9	15	10	15
74LS02	NOR	10	15	10	15
74LS08	AND	8	15	10	20
74LS32	OR	14	22	14	22

EJEMPLO 2.36

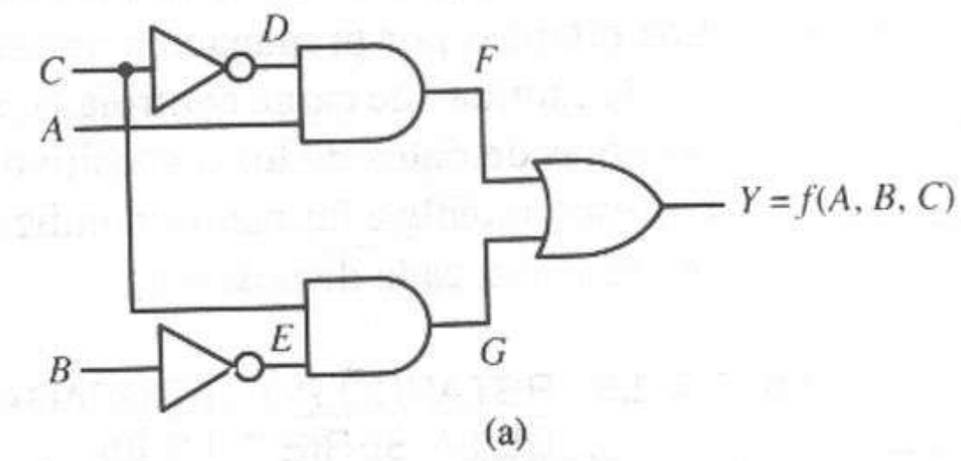
Se aplica una serie de entradas al circuito de la figura 2.25a, con lo cual se obtiene el diagrama de tiempos de la figura 2.25b. Cada compuerta tiene un retardo por propagación t_{PD} de una unidad de tiempo. Queremos determinar la tabla de verdad y la expresión de conmutación mínima para este circuito.

Partimos del diagrama de tiempos y obtenemos la tabla de verdad de la figura 2.25c examinando las salidas de cada compuerta después de cada uno de los cambios en la entrada. Como las señales tardan diferentes cantidades de tiempo en propagarse hasta la salida de la compuerta, debemos esperar hasta que la propagación de la señal termine antes de determinar la salida correspondiente a la entrada actual. Observe que la señal no se propaga a través de más de tres compuertas, por lo que no transcurren más de tres unidades de tiempo entre un cambio en la entrada y una salida estable.

Por ejemplo, en el instante t_1, la entrada C cambia de 0 a 1, lo que hace que la salida D del inversor y la salida G de la compuerta AND cambien en el instante $t_1 + 1$. El cambio en G hace que la salida Y de la compuerta OR cambie de 0 a 1 en el instante $t_1 + 2$. Así, el cambio en la entrada necesitó dos unidades de tiempo para propagarse de la entrada del circuito C a la salida del circuito Y. Por tanto, debemos esperar un tiempo de $t_1 + 2$ para determinar el valor final de Y. Observe que el cambio de entrada en el instante t_2 también requirió dos unidades de tiempo para propagarse hasta la salida, mientras que los cambios en los instantes t_4 y t_7 requieren tres unidades de tiempo.

Utilizamos la tabla de verdad para escribir la lista de mintérminos y obtener una expresión mínima de conmutación de la manera siguiente:

$$\begin{aligned} f(A, B, C) &= \sum m(1, 4, 5, 6) \\ &= \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} \\ &= A\bar{C} + \bar{B}C \end{aligned}$$



ABC	f(A, B, C)
000	0
001	1
010	0
011	0
100	1
101	1
110	1
111	0

(c)

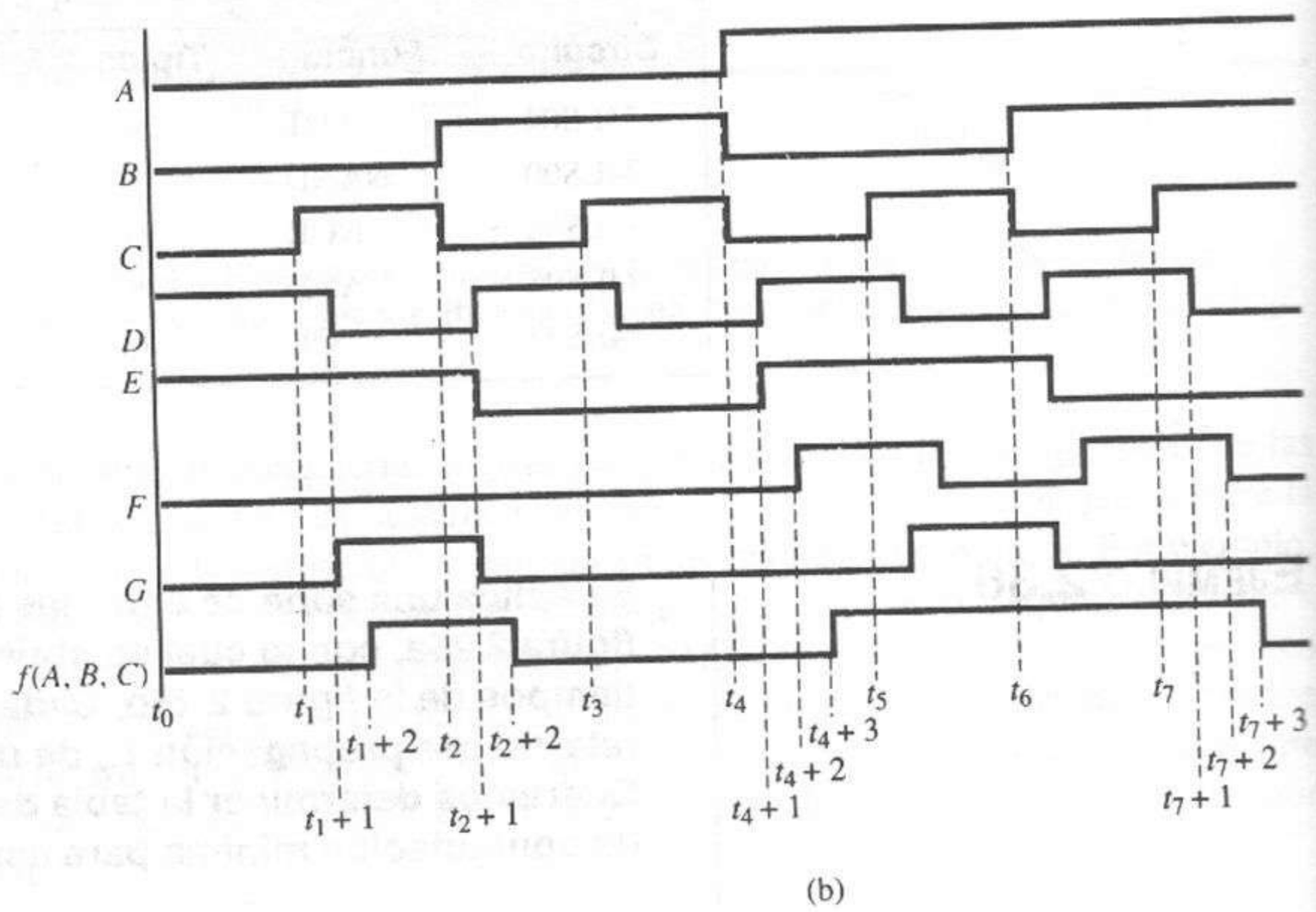


Figura 2.25 Obtención de una tabla de verdad a partir de un diagrama de tiempos. **(a)** Diagrama de circuitos. **(b)** Diagrama de tiempos. **(c)** Tabla de verdad.

2.5 Síntesis de circuitos lógicos combinatorios

Hasta ahora, hemos presentado varias herramientas que pueden servir para analizar y sintetizar redes de conmutación. Estas herramientas incluyen al álgebra de conmutación, los dispositivos de conmutación, las tablas de verdad y los diagramas de tiempos. En esta sección utilizaremos algunas de estas herramientas para el diseño e implantación de redes de conmutación.

2.5.1 Redes AND-OR y NAND

Una red AND-OR utiliza compuertas AND para formar términos producto y una compuerta OR para formar la suma de estos productos. Por tanto, una red

de conmutación que deba implantarse en lógica AND-OR debe expresarse en forma de suma de productos (SOP). Por ejemplo, podemos implantar la función

$$f_\delta(p, q, r, s) = p\bar{r} + qrs + \bar{p}s$$

de manera directa en la lógica AND-OR como se muestra en la figura 2.26a.

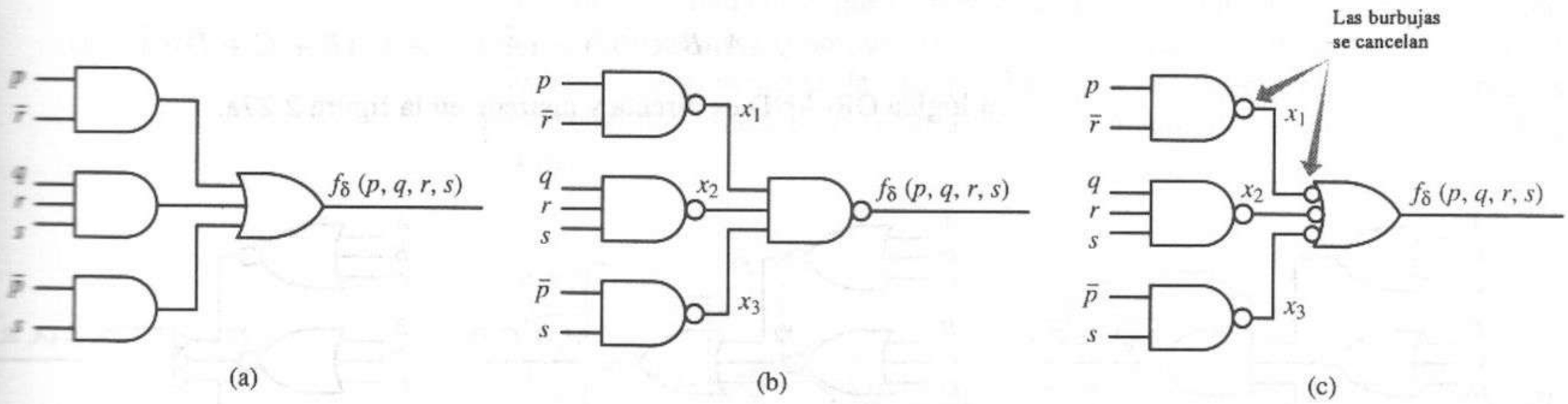


Figura 2.26 Implantaciones de $f_\delta(p, q, r, s) = p\bar{r} + qrs + \bar{p}s$. **(a)** Red AND-OR. **(b)** Red NAND. **(c)** Red NAND (forma preferida).

Podemos usar una traducción sencilla con álgebra de conmutación para transformar una expresión de suma de productos en una forma adecuada para la implantación NAND directa. Colocamos dos barras sobre toda la función SOP, y utilizamos el teorema de DeMorgan (teorema 8) para determinar la forma NAND de la función.

$$\begin{aligned} f_\delta(p, q, r, s) &= \overline{\overline{p\bar{r} + qrs + \bar{p}s}} && [T3] \\ &= \overline{\overline{p\bar{r}} \cdot \overline{qrs} \cdot \overline{\bar{p}s}} && [T8(a)] \\ &= \overline{x_1 \cdot x_2 \cdot x_3} \end{aligned}$$

donde $x_1 = p\bar{r}$, $x_2 = \overline{qrs}$, y $x_3 = \bar{p}s$. En la figura 2.26b mostramos la realización NAND de la función.

La figura 2.26c presenta el mismo circuito, pero con la compuerta NAND de la salida en su forma equivalente de DeMorgan. Si escribimos la expresión lógica para la salida,

$$\begin{aligned} f_\delta(p, q, r, s) &= \bar{x}_1 + \bar{x}_2 + \bar{x}_3 \\ &= \overline{\overline{p\bar{r}}} + \overline{\overline{qrs}} + \overline{\overline{\bar{p}s}} \\ &= p\bar{r} + qrs + \bar{p}s \end{aligned}$$

Observe que las burbujas de inversión en ambos extremos de las líneas x_1, x_2, y x_3 se cancelan, lo que hace que el diagrama recuerde a la figura 2.26a y así ilustre de manera clara el hecho de que se está implantando una forma de suma de productos. Por ello, el formato de la figura 2.26c es preferible al de la figura 2.26b cuando se usa un circuito NAND.

Las técnicas aquí utilizadas pueden aplicarse a cualquier función de suma de productos para obtener una red AND-OR o NAND.

2.5.2 Redes OR-AND y NOR

Una red OR-AND utiliza compuertas OR para formar términos de suma y una compuerta AND para formar el producto de estas sumas. En consecuencia, una función de conmutación dada que vaya a implantarse en lógica OR-AND deberá expresarse en forma de producto de sumas (POS). Por ejemplo, la realización de la función

$$f_\epsilon(A, B, C, D) = (\bar{A} + B + C)(B + C + D)(\bar{A} + D)$$

en lógica OR-AND es directa y aparece en la figura 2.27a.

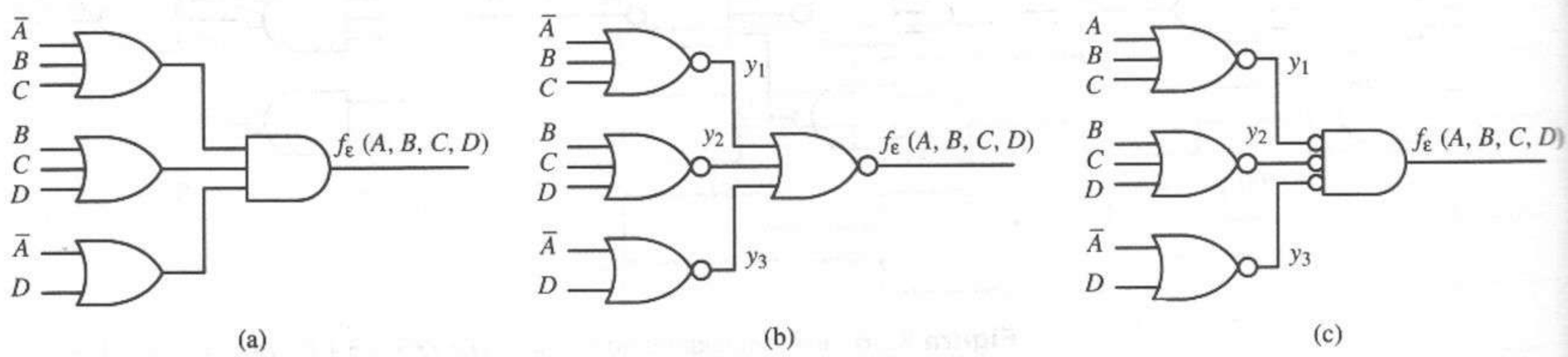


Figura 2.27 Implantaciones de $f_\epsilon(A, B, C, D) = (\bar{A} + B + C)(B + C + D)(\bar{A} + D)$. **(a)** Red OR-AND. **(b)** Red NOR. **(c)** Red NOR (forma preferible).

Podemos aprovechar la misma transformación del álgebra de conmutación que utilizamos con anterioridad y expresar $f_\epsilon(A, B, C, D)$ en una forma adecuada para la implantación NOR directa, mediante el teorema de DeMorgan:

$$\begin{aligned} f_\epsilon(A, B, C, D) &= \overline{\overline{(\bar{A} + B + C)(B + C + D)(\bar{A} + D)}} && \text{[T3]} \\ &= \overline{\overline{\bar{A} + B + C} + \overline{B + C + D} + \overline{\bar{A} + D}} && \text{[T8(b)]} \\ &= \overline{y_1 + y_2 + y_3} \end{aligned}$$

donde $y_1 = \overline{\bar{A} + B + C}$, $y_2 = \overline{B + C + D}$, y $y_3 = \overline{\bar{A} + D}$. La realización NOR de esta función aparece en la figura 2.27b.

La figura 2.27c muestra el mismo circuito, pero con la compuerta NOR de salida en su forma equivalente de DeMorgan. Si escribimos la expresión lógica de salida, tenemos

$$\begin{aligned} f_\epsilon(A, B, C, D) &= \bar{y}_1 \cdot \bar{y}_2 \cdot \bar{y}_3 \\ &= \overline{\overline{(\bar{A} + B + C)}} \cdot \overline{\overline{(B + C + D)}} \cdot \overline{\overline{(\bar{A} + D)}} \\ &= (\bar{A} + B + C)(B + C + D)(\bar{A} + D) \end{aligned}$$

Como en el caso de los circuitos NAND de dos niveles, las burbujas de inversión se cancelan. Este formato aclara que se está implantando una función en forma de producto de sumas y por ello es la forma preferible para trazar un circuito NOR de dos niveles.

Podemos generalizar el método anterior para implantar cualquier función de producto de sumas en lógica NOR.

2.5.3 Circuitos de dos niveles

Las redes con una estructura similar a la de las figuras 2.26 y 2.27 se conocen como *redes de dos niveles*. Las señales de entrada deben pasar a través de dos niveles de compuertas antes de llegar a la salida. Como muestra la figura 2.28a, definimos el primer nivel como el nivel que contiene la compuerta que produce la salida. Las compuertas que reciben las entradas del circuito están en el segundo nivel. Si se requieren compuertas NOT en las líneas de entrada, se tiene una red de tres niveles, como en la figura 2.28b. Una red tiene n niveles si al menos una señal de entrada debe pasar a través de n compuertas antes de llegar a la salida.

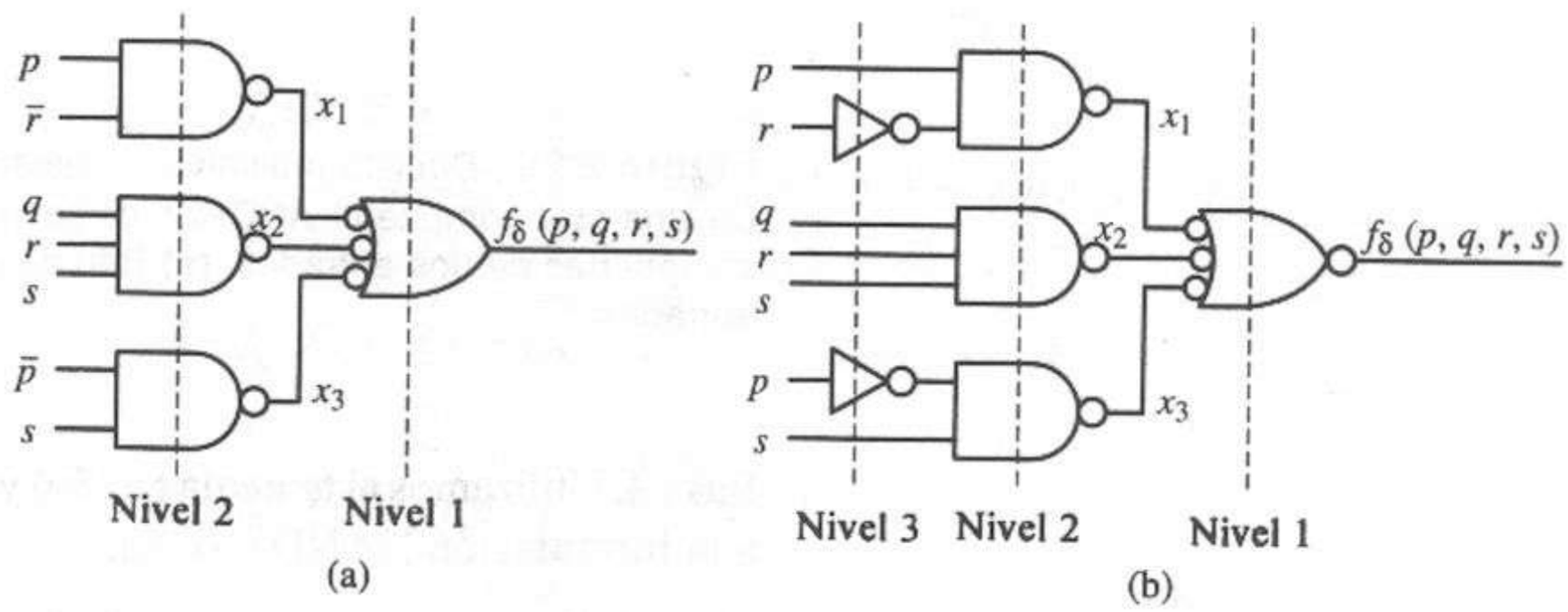


Figura 2.28 Estructuras de red de dos y tres niveles. **(a)** Red de dos niveles. **(b)** Red de tres niveles.

Las funciones de conmutación en la forma SOP o POS tienen una implantación directa en las redes de dos niveles si se dispone de las entradas en forma complementada y no complementada. Se necesita una red de tres niveles cuando sólo se dispone de una forma de las entradas. En este caso, sólo se necesitan compuertas NOT en el tercer nivel.

Con frecuencia hay que crear circuitos con más de dos niveles cuando existen límites para el *fan-in* de las compuertas. Por ejemplo, la función $f(a, b, c, d, e) = abcde$ se puede realizar con una única compuerta AND de cinco entradas, como se muestra en la figura 2.29a, pero si el diseñador tiene la restricción de trabajar sólo con compuertas AND de dos entradas, necesitará un circuito de tres o cuatro niveles, como los de las figuras 2.29b y c, respectivamente. El lector debe verificar que ambos circuitos son equivalentes.

Ahora, el lector dispone de todas las herramientas necesarias para tomar una función de conmutación expresada en forma de lista de mintérminos o maxtérminos e implantarla en lógica NAND y NOR, respectivamente. A continuación bosquejamos el proceso de implantación para la lógica NAND; los términos entre paréntesis se usarán si la implantación debe hacerse en lógica NOR.

Paso 1. Expresamos la función en forma de lista de mintérminos (maxtérminos).

Paso 2. Escribimos los mintérminos (maxtérminos) en forma algebraica.

Paso 3. Simplificamos la función en la forma de suma de productos (producto de sumas) mediante el álgebra de conmutación.

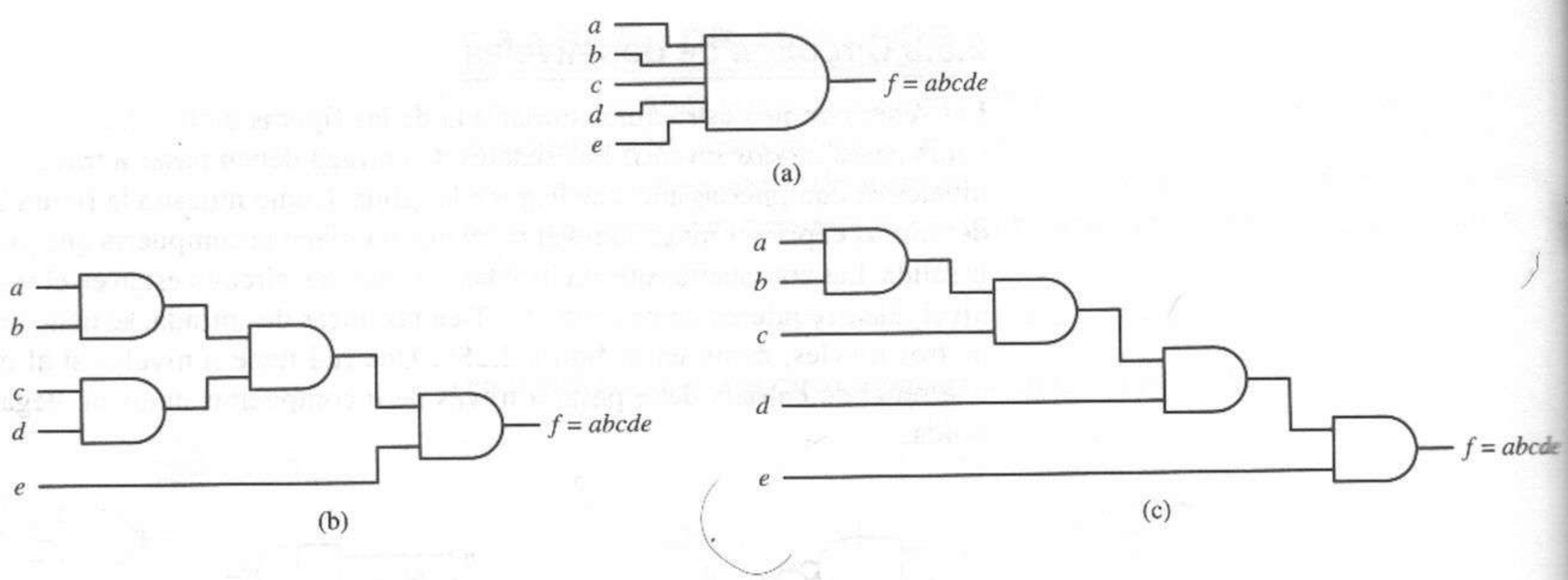


Figura 2.29 Circuito multinivel necesario debido a las restricciones de *fan-in*. **(a)** Una sola compuerta AND con cinco entradas. **(b)** Red de tres niveles con compuertas de dos entradas. **(c)** Red de cuatro niveles de compuertas de dos entradas.

Paso 4. Utilizamos el teorema 8a (8b) y el teorema 3 para transformar la expresión a la formulación NAND (NOR).

Paso 5. Trazamos el diagrama lógico NAND (NOR).

Ahora ilustraremos este proceso para la función $f_{\varnothing}(X, Y, Z) = \sum m(0, 3, 4, 5, 7)$

EJEMPLO 2.37

Implantar $f_{\varnothing}(X, Y, Z) = \sum m(0, 3, 4, 5, 7)$ en lógica NAND.

1. $f_{\phi}(X, Y, Z) = \sum m(0, 3, 4, 5, 7)$
2. $f_{\phi}(X, Y, Z) = m_0 + m_3 + m_4 + m_5 + m_7$
   $= \bar{X}\bar{Y}\bar{Z} + \bar{X}YZ + X\bar{Y}\bar{Z} + X\bar{Y}Z + XYZ$
3. $f_{\phi}(X, Y, Z) = \bar{Y}\bar{Z} + YZ + XZ$ [T6(a)]

4a. $f_{\phi}(X, Y, Z) = \overline{\overline{\bar{Y}\bar{Z}}} + \overline{\overline{YZ}} + \overline{\overline{XZ}}$ [T3]

o bien

4b. $f_{\phi}(X, Y, Z) = \overline{\overline{\bar{Y}\bar{Z} + YZ + XZ}}$ [T3]
$= \overline{\overline{\bar{Y}\bar{Z}} \cdot \overline{YZ} \cdot \overline{XZ}}$ [T8(a)]

Obtenemos el diagrama lógico de la figura 2.30a a partir de la expresión del paso 4a, y decimos que es una *realización SOP mínima de dos niveles* de la función de conmutación. Este ejemplo ilustra de manera completa el proceso de diseño.

EJEMPLO 2.38

Implantar $f_{\varnothing}(X, Y, Z) = \sum m(0, 3, 4, 5, 7)$ en lógica NOR.

1. $f_{\phi}(X, Y, Z) = \prod M(1, 2, 6)$

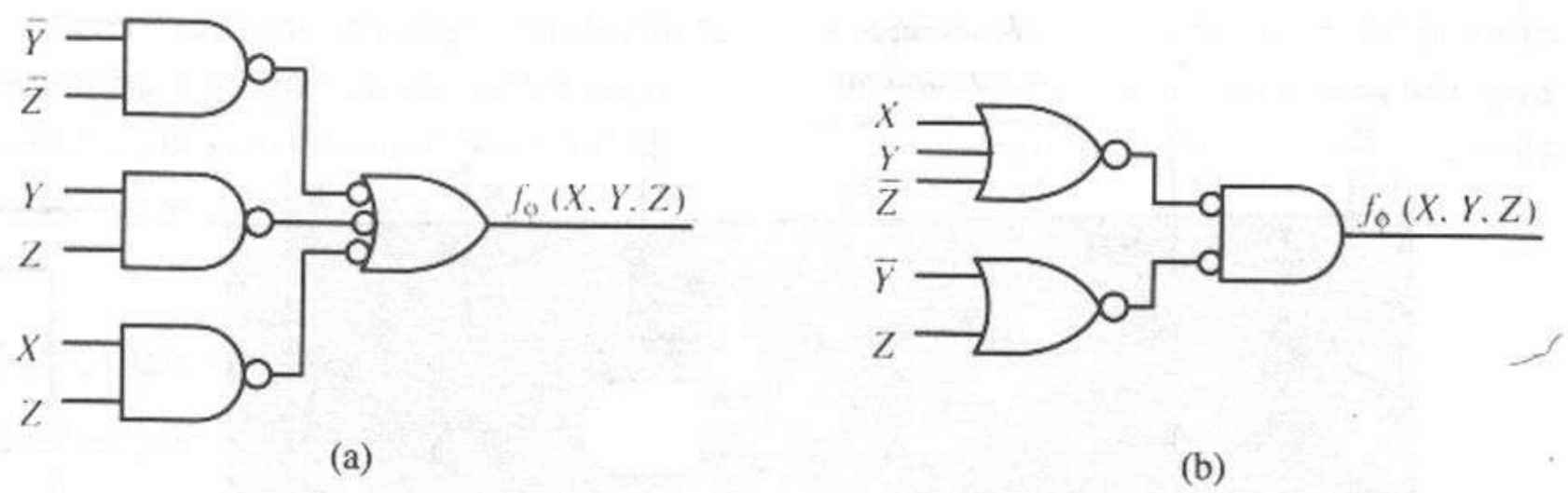


Figura 2.30 Forma canónica de $f_\varnothing(X, Y, Z) = \sum m(0, 3, 4, 5, 7)$ en redes NAND y NOR. **(a)** Realización NAND. **(b)** Realización NOR.

$$
\begin{aligned}
2.\ f_\phi(X, Y, Z) &= M_1 \cdot M_2 \cdot M_6 \\
&= (X + Y + \bar{Z})(X + \bar{Y} + Z)(\bar{X} + \bar{Y} + Z) \\
3.\ f_\phi(X, Y, Z) &= (X + Y + \bar{Z})(\bar{Y} + Z) && \text{[T6(b)]} \\
4a.\ f_\phi(X, Y, Z) &= \overline{\overline{(X + Y + \bar{Z})}} \cdot \overline{\overline{(\bar{Y} + Z)}} && \text{[T3]} \\
&\text{o} \\
4b.\ f_\phi(X, Y, Z) &= \overline{\overline{(X + Y + \bar{Z})(\bar{Y} + Z)}} && \text{[T3]} \\
&= \overline{\overline{(X + Y + \bar{Z})}} + \overline{\overline{(\bar{Y} + Z)}} && \text{[T8(b)]}
\end{aligned}
$$

La red NOR obtenida del paso 4a. aparece en la figura 2.30b y es una *realización POS mínima de dos niveles* de la función de conmutación. Ambas redes de la figura 2.30 son implantaciones de la función $f_\varnothing(X, Y, Z)$.

2.5.4 Circuitos AND-OR-inversor

Un circuito AND-OR-inversor (AOI) está formado por un conjunto de compuertas AND, cuyas salidas se alimentan a una compuerta NOR y, por tanto, puede servir para realizar con facilidad los circuitos de suma de productos de dos niveles. En la figura 2.31 aparece una configuración típica, similar a la utilizada en la lógica estándar (7454) de la serie 7400. En general, definimos los circuitos según el número de entradas a las compuertas AND. Por ejemplo, un circuito con tres compuertas AND en el que una tiene dos entradas, otra tres entradas y la última cuatro, se designa como circuito AOI 2-3-4.

El circuito 7454 que aparece en la figura 2.31 es un circuito AOI 2-2-2-2 que realiza la función

$$F = \overline{AB + CD + EF + GH}$$

Aunque podemos utilizar el circuito AOI de varias formas, sólo ilustraremos aquí una aplicación. Si B, D, F y H funcionan como líneas de activación (control) y A, C, E y G son líneas de información, podemos usar el circuito anterior para reunir la información de las cuatro fuentes A, C, E y G en un único canal (este circuito es un *multiplexor 4 a 1*, y se describirá con detalle en el capítulo 4). Por ejemplo, si $A = Y1$, $C = Y2$, $E = Y3$ y $G = Y4$, y $B = 1$ y $D = F = H = 0$, la salida será $F = \overline{Y1}$. Si entonces cambiamos las líneas de activación de modo que $D = 1$ y $B = F = H = 0$, la salida será $F = \overline{Y2}$. Así, al establecer las señales de las líneas de activación en 1 lógico de la manera ilustrada, podemos

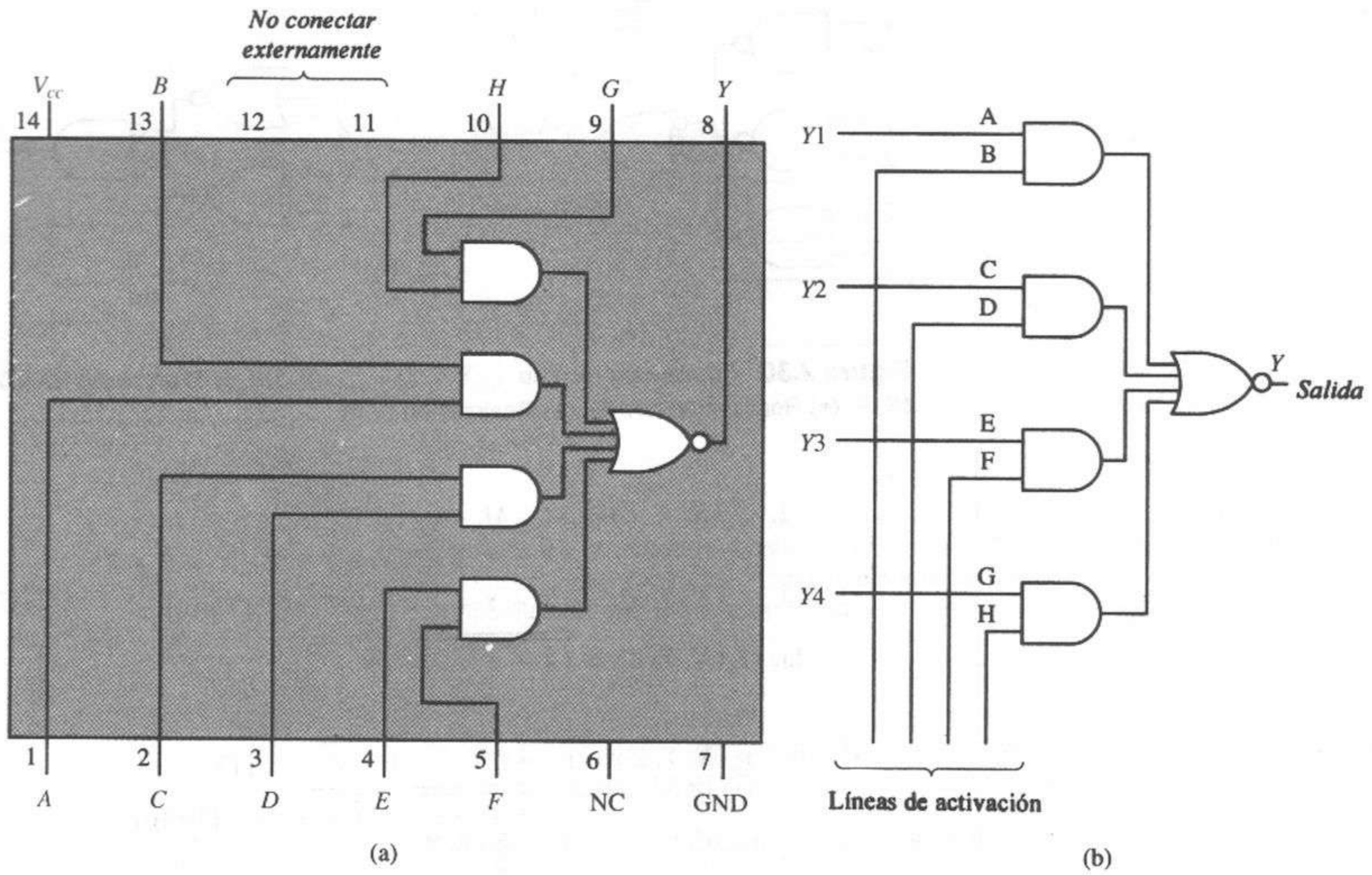


Figura 2.31 Circuito AND-OR-inversor 7454 2-2-2-2. **(a)** Paquete de circuito 7454 (vista superior). **(b)** 7454 utilizado como un multiplexor 4 a 1.

reunir un conjunto de flujos de datos representados por A, C, E y G en un único flujo representado por F.

2.5.5 Factorización

La factorización es una técnica que nos permite obtener formas de nivel superior para las funciones de conmutación (las cuales requieren circuitos en los que las señales se pueden propagar a través de más de dos niveles de compuertas lógicas). La importancia de estas formas superiores surge del hecho de que muchas veces, necesitan menos hardware y, por tanto, su implantación es más económica. También requerimos formas de nivel superior en situaciones en las que debemos utilizar *fan-in* limitado. En estos casos nos valemos de la factorización para reducir la cantidad de literales en términos grandes de productos o sumas hasta obtener valores menores o iguales a la cantidad disponible de entradas de compuerta. Sin embargo, las formas de nivel superior son más difíciles de diseñar que las formas SOP o POS sencillas y, por lo general, son más lentas por que tienen más de dos niveles de compuertas lógicas.

La factorización, que por lo general implica el uso de la ley distributiva (postulado 5) del álgebra de conmutación, es esencialmente un arte en el que la

experiencia tiene un papel importante. La técnica se complica aún más porque debemos agregar cierta redundancia en un paso intermedio para obtener una realización sencilla mediante factorización. Demostraremos este método a través de los ejemplos siguientes.

EJEMPLO 2.39

Suponga la siguiente función de cuatro variables:

$$f_\lambda(A,B,C,D) = A\bar{B} + A\bar{D} + A\bar{C}$$

La realización de dos niveles de esta función usando compuertas NAND aparece en la figura 2.32a.

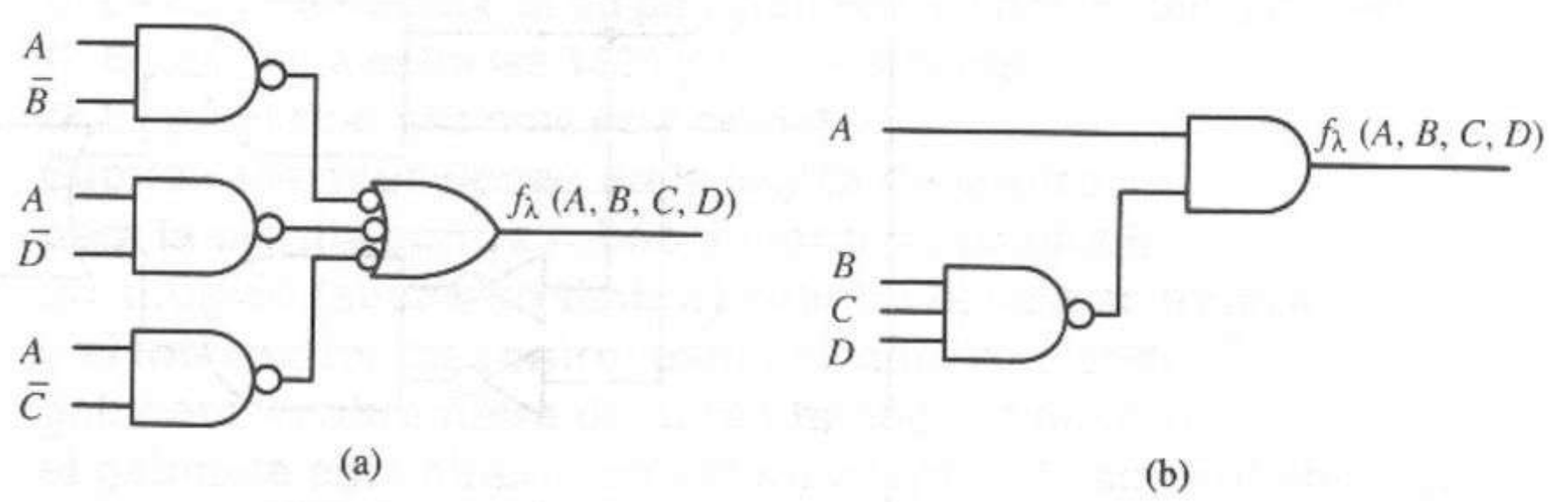


Figura 2.32 Realizaciones de $f_\lambda(A, B, C, D)$. **(a)** Forma original. **(b)** Después de la factorización.

Observe que esta realización de segundo orden de la función requiere cuatro compuertas y nueve entradas de compuerta. Sin embargo, si aplicamos factorización a la función, podemos obtener una realización de orden superior, de la manera siguiente:

$$\begin{aligned} f_\lambda(A, B, C, D) &= A\bar{B} + A\bar{D} + A\bar{C} \\ &= A(\bar{B} + \bar{D} + \bar{C}) \\ &= A(\overline{BCD}) \end{aligned}$$

Esta realización de $f_\lambda(A, B, C, D)$, que se muestra en la figura 2.32b, sólo requiere dos compuertas y cinco entradas de compuerta.

EJEMPLO 2.40

Realizar la función $f(a, b, c, d) = \sum m(8, 13)$ utilizando sólo compuertas AND y OR de dos entradas.

Comenzamos escribiendo la forma SOP canónica:

$$\begin{aligned} f(a, b, c, d) &= \sum m(8, 13) \\ &= a\bar{b}\bar{c}\bar{d} + ab\bar{c}d \end{aligned} \tag{2.34}$$

No podemos reducir los dos términos producto de la ecuación 2.34 mediante el álgebra de conmutación. En consecuencia, se necesitarían dos compuertas AND de cuatro entradas y una compuerta OR de dos entradas para realizar un circuito AND-OR de dos niveles.

Puesto que sólo disponemos de compuertas de dos entradas, podemos aplicar factorización para reducir el tamaño de los términos producto, como sigue:

$$\begin{aligned} f(a, b, c, d) &= a\bar{b}\bar{c}\bar{d} + ab\bar{c}d \\ &= (a\bar{c})(bd + \bar{b}\bar{d}) \end{aligned} \tag{2.35}$$

En la ecuación 2.35, ningún término de producto o suma contiene más de dos literales. Así, podemos realizar esta expresión de conmutación con compuertas de dos entradas, como se muestra en la figura 2.33. Observe que el circuito contiene cuatro niveles de compuertas lógicas, incluidos los inversores de las entradas.

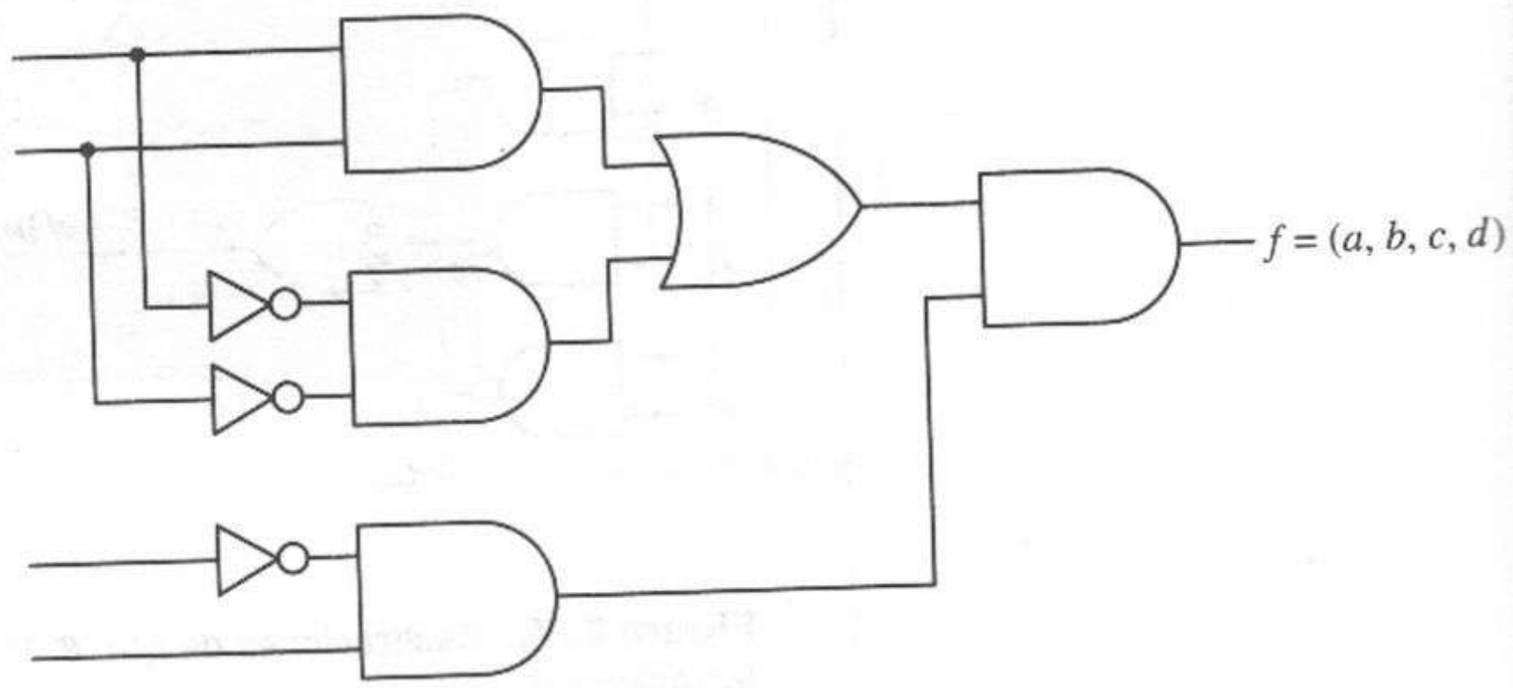


Figura 2.33 Uso de la factorización utilizada para realizar $f(a, b, c, d) = \sum m(8, 13)$ con compuertas de dos entradas.

La estrategia algebraica que aplicamos en este ejemplo se puede realizar también mediante varios métodos gráficos y tabulares, como el mapa de Karnaugh (mapa K), que presentaremos en el capítulo 3. Si desea más detalles sobre el tema, el lector debe consultar [3].

2.6 Aplicaciones

Hasta ahora hemos presentado varias herramientas, como el álgebra de conmutación, las tablas de verdad y los diagramas de Venn, fundamentales para el análisis y la síntesis de las redes lógicas. Además, hemos presentado los conceptos básicos del análisis y diseño de un circuito lógico digital. En los siguientes ejemplos ilustramos el uso de estas técnicas para resolver problemas.

Algunas áreas en que las herramientas básicas tienen amplia aplicación son las de la lógica simbólica y las funciones de verdad. No consideraremos estos temas con detalle, sino que ilustraremos el uso de la lógica en varios ejemplos muy sencillos.

EJEMPLO 2.41

Se diseña una alarma contra robos para un banco, de modo que percibe cuatro líneas de señal como entradas. La línea *A* es del interruptor secreto de control, la línea *B* es de un sensor de presión debajo de una caja fuerte que se encuentra en un gabinete cerrado, la línea *C* es de un reloj alimentado por baterías, y la línea *D* se conecta a un interruptor en la puerta del gabinete cerrado. Las siguientes condiciones producen un voltaje de 1 lógico en cada línea:

***A*: El interruptor de control está cerrado.**
***B*: La caja fuerte está en su posición normal dentro del gabinete.**
***C*: El reloj está entre las 1000 y las 1400 horas.**
***D*: La puerta del gabinete está cerrada.**

Escribir las ecuaciones de la lógica de control para la alarma contra robos a modo de producir un 1 lógico (suena un timbre) cuando la caja se mueve y el interruptor de control está cerrado, o cuando el gabinete se abre fuera de horas hábiles, o cuando el gabinete está abierto con el interruptor de control abierto.

El enunciado "cuando la caja se mueve y el interruptor de control está cerrado" se representa mediante $A\overline{B}$. "Cuando el gabinete se abre fuera de horas hábiles" se representa mediante $\overline{C}\overline{D}$. "Cuando el gabinete está abierto con el interruptor de control abierto" se representa mediante $\overline{A}\overline{D}$. Por tanto, la ecuación lógica para la alarma contra robos es

$$f(A, B, C, D) = A\bar{B} + \bar{C}\bar{D} + \bar{A}\bar{D}$$

EJEMPLO 2.42

Juan y María Pérez tienen dos hijos, José y Susana. Cuando salen a comer, van a un restorán que sólo sirve hamburguesas o a uno que sólo sirve pollo. Antes de salir, la familia vota para elegir el restorán. Gana la mayoría, excepto cuando los papás están de acuerdo, en cuyo caso ellos ganan. Cualquier otro empate implica ir al restorán de pollo. Diseñar un circuito lógico que seleccione en forma automática el restorán elegido cuando toda la familia vota.

Si 1 representa un voto por las hamburguesas y 0 un voto por el pollo, la tabla de verdad para el circuito de votación es la de la tabla 2.9. La función lógica es

$$\begin{aligned} f(A, B, C, D) = {} & \bar{A}BCD + A\bar{B}CD + AB\bar{C}\bar{D} \\ & + AB\bar{C}D + ABC\bar{D} + ABCD \end{aligned}$$

$$= \bar{A}BCD + A\bar{B}CD + AB$$
$$= AB + ACD + \bar{A}BCD$$
$$= AB + ACD + BCD$$

La figura 2.34 muestra el circuito lógico para esta función.

TABLA 2.9 TABLA DE VERDAD PARA LA VOTACIÓN DE LA FAMILIA PÉREZ

Juan A	María B	José C	Susana D	Voto por las hamburguesas f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

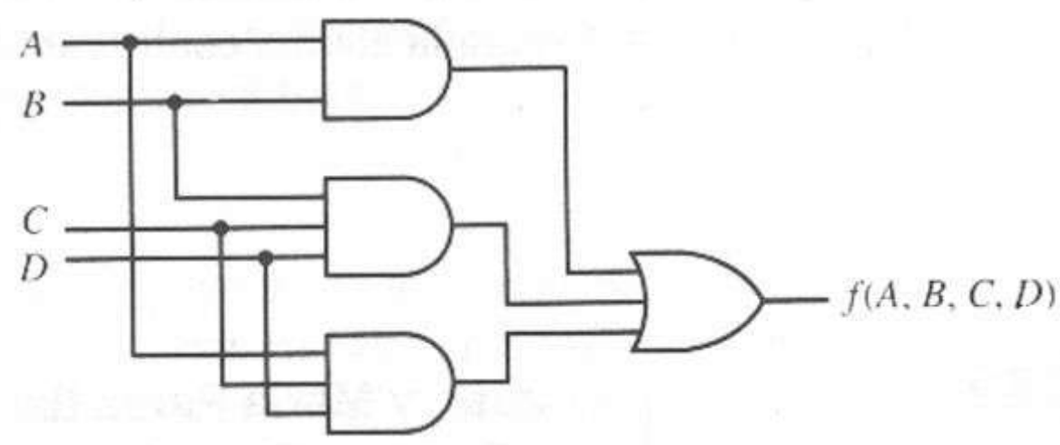


Figura 2.34 Circuito lógico para el ejemplo de la votación por el restorán.

EJEMPLO 2.43

Deducir las ecuaciones lógicas para un circuito que sume los dos números binarios de 2 bits $(A_1A_0)_2$ y $(B_1B_0)_2$ y produzca los bits de suma $(S_1S_0)_2$ y el bit de acarreo de salida C_1; es decir,

$$\begin{array}{r} A_1A_0 \\ +\,B_1B_0 \\ \hline C_1S_1S_0 \end{array}$$

Tenemos cuatro entradas, A_1, A_0, B_1 y B_0, y tres salidas, C_1, S_1 y S_0, la tabla de verdad es entonces como se muestra en la tabla 2.10. Deducimos cada fila de la

TABLA 2.10 TABLA DE VERDAD PARA EL SUMADOR BINARIO DE 2 BITS

	A_1	A_0	B_1	B_0	C_1	S_1	S_0
	0	0	0	0	0	0	0
	0	0	0	1	0	0	1
	0	0	1	0	0	1	0
	0	0	1	1	0	1	1
	0	1	0	0	0	0	1
	0	1	0	1	0	1	0
	0	1	1	0	0	1	1
→	0	1	1	1	1	0	0
	1	0	0	0	0	1	0
	1	0	0	1	0	1	1
	1	0	1	0	1	0	0
	1	0	1	1	1	0	1
	1	1	0	0	0	1	1
	1	1	0	1	1	0	0
	1	1	1	0	1	0	1
	1	1	1	1	1	1	0

tabla de verdad como se muestra en el siguiente cálculo de la octava fila.

$$\begin{array}{rrr} 1 & 1 & \\ & 0 & 1 \\ + & 1 & 1 \\ \hline 1 & 0 & 0 \end{array}$$

Obtenemos la siguiente ecuación lógica para los términos C_1, S_1 y S_0 a partir de la tabla de verdad:

$$\begin{aligned} S_0 = {} & \bar{A}_1\bar{A}_0\bar{B}_1B_0 + \bar{A}_1\bar{A}_0B_1B_0 + \bar{A}_1A_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0B_1\bar{B}_0 \\ & + A_1\bar{A}_0\bar{B}_1B_0 + A_1\bar{A}_0B_1B_0 + A_1A_0\bar{B}_1\bar{B}_0 + A_1A_0B_1\bar{B}_0 \end{aligned}$$

$$\begin{aligned} S_1 = {} & \bar{A}_1\bar{A}_0B_1\bar{B}_0 + \bar{A}_1\bar{A}_0B_1B_0 + \bar{A}_1A_0\bar{B}_1B_0 + \bar{A}_1A_0B_1\bar{B}_0 \\ & + A_1\bar{A}_0\bar{B}_1\bar{B}_0 + A_1\bar{A}_0\bar{B}_1B_0 + A_1A_0\bar{B}_1\bar{B}_0 + A_1A_0B_1B_0 \end{aligned}$$

$$\begin{aligned} C_1 = {} & \bar{A}_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0 + A_1\bar{A}_0B_1B_0 + A_1A_0\bar{B}_1B_0 \\ & + A_1A_0B_1\bar{B}_0 + A_1A_0B_1B_0 \end{aligned}$$

Podemos reducir estas expresiones a:

$$S_0 = A_0\bar{B}_0 + \bar{A}_0B_0$$

$$S_1 = \bar{A}_1\bar{A}_0B_1 + \bar{A}_1B_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{B}_1\bar{B}_0 + A_1\bar{A}_0\bar{B}_1$$

$$C_1 = A_0B_1B_0 + A_1A_0B_0 + A_1B_1$$

2.7 Diseño de circuitos lógicos asistido por computadora

2.7.1 El ciclo de diseño

Una gran parte de los circuitos lógicos digitales de la actualidad contienen el equivalente de miles o cientos de miles de compuertas lógicas. La mayor parte de estos circuitos se fabrican en chips de circuitos integrados (IC). El diseño y fabricación de un circuito VLSI (integración a muy grande escala) es un proceso caro y complejo. Por tanto, es necesario verificar que el diseño del circuito lógico sea correcto antes de iniciar el diseño del circuito real, para tener una alta probabilidad de operación correcta desde que se inicie la fabricación del circuito. Esto también es cierto al diseñar sistemas digitales con varios IC y tarjetas de circuitos. Es un hecho que no se pueden desarrollar ni verificar los circuitos y sistemas de tal complejidad sin el apoyo de herramientas de diseño asistido por computadora (CAD).

El ciclo de diseño para un circuito lógico digital abarca varios pasos entre el concepto y la implantación física, incluida la síntesis de diseño, la simulación, la realización y la prueba. La figura 2.35 describe este proceso. A partir del enunciado del problema, el diseñador comienza por desarrollar una solución abstracta que se va transformando sistemáticamente en un circuito lógico digital. Esta transformación es apoyada por el modelado y la evaluación del circuito en cada nivel de abstracción del diseño. Evaluamos un diseño mediante un modelo que simula su operación, lo que permite verificar la respuesta del circuito a diversos estímulos de entrada. El modelo se revisa y simula cuantas veces sea necesario hasta obtener las respuestas correctas. Además de verificar la correcta operación, podemos evaluar los efectos de las diferentes opciones de diseño sobre el desempeño del circuito como apoyo para la toma de decisiones de diseño con un costo realista. Una vez aceptado el comportamiento modelado del diseño, se desarrolla e implanta el diseño físico. Por último, se prueba el circuito terminado, y los resultados de la prueba se comparan con el comportamiento modelado para detectar dispositivos con fallas.

En este capítulo describiremos los procesos y herramientas de CAD empleados en las fases de síntesis y análisis del diseño de circuitos digitales. Examinaremos cada bloque de las fases de síntesis y análisis de la figura 2.35. En primer lugar, analizaremos el modelado del diseño. En la siguiente sección describiremos la captura de esquemas y otras herramientas de CAD que capturan los modelos de circuitos lógicos y los traducen al formato de la base de datos de diseño. Después analizaremos la simulación lógica digital, que se aplica para verificar el comportamiento lógico y los tiempos de un diseño. Por último, examinaremos las herramientas de CAD que deducen expresiones de conmutación mínimas para las funciones lógicas.

2.7.2 Modelado de un circuito digital

El modelado de un circuito lógico o sistema digital tiene varios propósitos. En primer lugar, el proceso de desarrollo de un modelo ayuda al diseñador a formalizar una solución. En segundo lugar, una computadora digital puede procesar un modelo de circuito para verificar que no tenga errores de diseño y sea correcto, y para predecir las características de tiempos. Además, se dispone

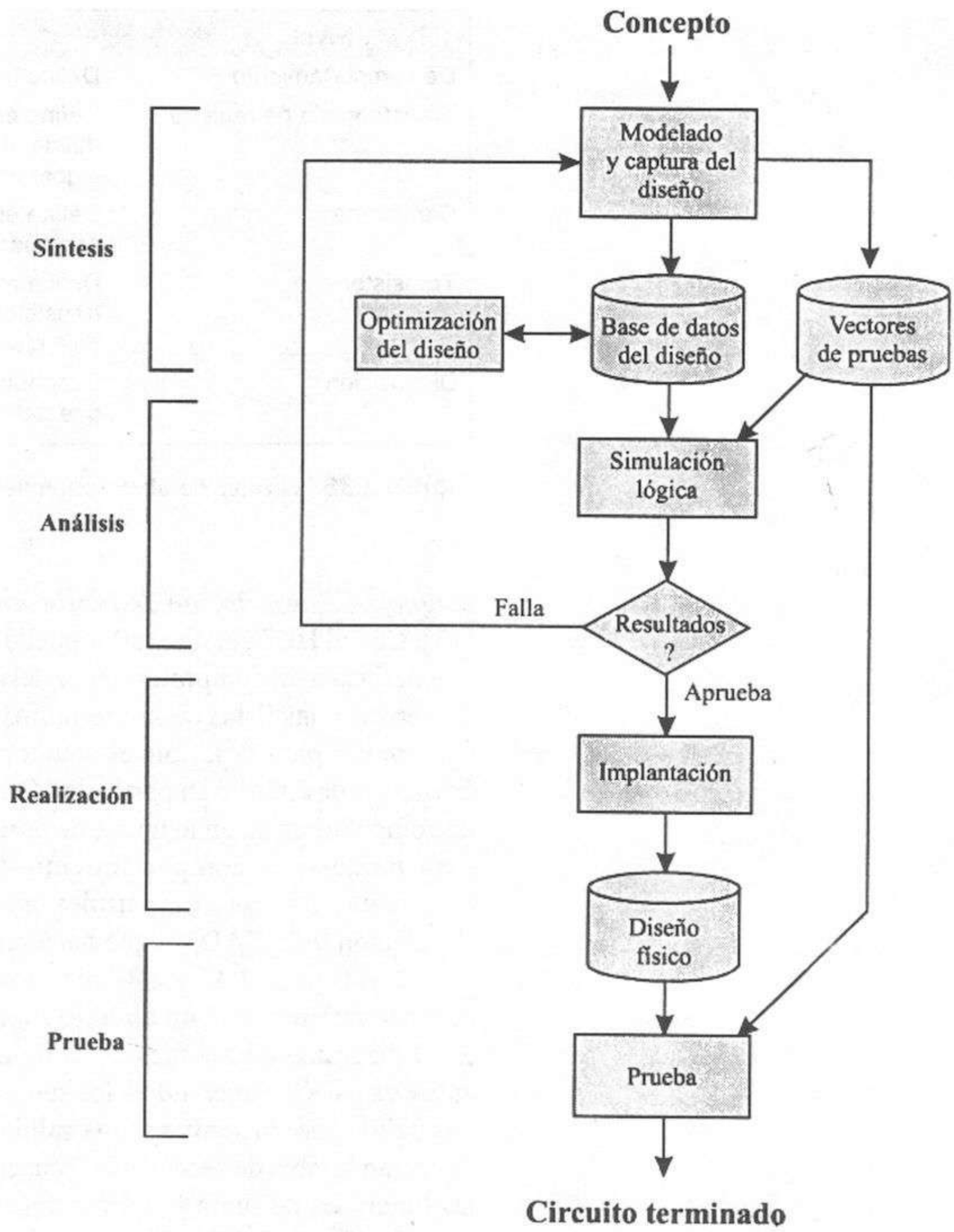


Figura 2.35 Diseño asistido por computadora de un circuito lógico digital.

de varias herramientas de CAD para realizar de manera automática todos o algunos de los pasos de síntesis, incluidas la optimización y transformación del diseño de una forma abstracta a una realización física.

Un modelo puede representar un sistema digital con diversos niveles de abstracción del diseño, que van desde su comportamiento hasta su estructura. Esto se ilustra en la figura 2.36. Con frecuencia, el diseñador comienza con un modelo abstracto de alto nivel que sólo describe el *comportamiento* deseado del circuito, sin especificar las componentes reales del mismo. Esto permite deducir las características esenciales del diseño, sin tener que ocuparse de los detalles de la implantación, y también proporciona una descripción comprensible del diseño que puede servir para desarrollar, documentar y mantener el circuito. Los modelos de comportamiento se representan comúnmente empleando un

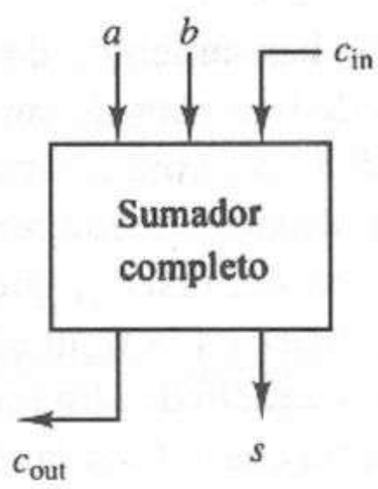
Nivel	Abstracción
De comportamiento	Define los algoritmos a realizar mediante el circuito.
Transferencia de registro	Define el circuito como una estructura de módulos y define el flujo de datos entre ellos, junto con el algoritmo de control.
Compuerta	Define el circuito como una estructura de compuertas lógicas primitivas.
Transistor	Define el circuito como una estructura de transistores y otros componentes electrónicos de bajo nivel.
Disposición	Describe los patrones geométricos de los materiales que definen el esquema físico del IC.

Figura 2.36 Niveles de abstracción del modelo para los circuitos lógicos digitales.

lenguaje de descripción de hardware (HDL), como VHDL [10-12] o Verilog [13]. Con el HDL, el diseñador puede expresar de manera formal los algoritmos que describen el comportamiento del circuito. Las ecuaciones lógicas, las tablas de verdad y las listas de mintérminos y maxtérminos son otros mecanismos de uso común para describir el comportamiento de un circuito sin implicar una estructura de circuito en particular. Otro método consiste en escribir un programa de computadora en un lenguaje de programación estándar, como C o FORTRAN, para modelar el comportamiento del circuito. Sin embargo, los modelos resultantes no son compatibles por lo general con otras herramientas de simulación y de CAD, lo que hace menos atractivo este método.

Las figuras 2.37 y 2.38 presentan tres formas diferentes de un modelo de comportamiento para un circuito sumador completo de 1 bit. Como se muestra en el diagrama de bloques de la figura 2.37a, un sumador completo tiene dos entradas para los operandos, los sumandos a y b, una entrada para el acarreo, c_{in}, una salida para la suma s y una salida para el acarreo c_{out}. Las figuras 2.37b y c muestran la tabla de verdad y las ecuaciones lógicas, respectivamente, que definen las funciones de suma y acarreo del sumador completo.

La figura 2.38 muestra un modelo de comportamiento VHDL para un sumador completo. Un modelo VHDL tiene dos partes: una entidad y una arquitectura. La *entidad* define la interfaz entre el circuito y el mundo exterior. La descripción de la entidad VHDL del sumador completo correspondiente al diagrama de bloques de la figura 2.37 enumera tres "puertos" de entrada, a, b y



(a)

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b)

$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = ab + ac_{in} + bc_{in}$$

(c)

Figura 2.37 Modelos de comportamiento para un circuito sumador completo. **(a)** Diagrama de bloques. **(b)** Tabla de verdad. **(c)** Ecuaciones lógicas.

```
entity full_adder is
     port(a, b, cin: in bit;
            a, cout: out bit)
end full_adder;

architecture behavior of full-adder is
begin
   process (a, b, cin)
       variable ai, bi, ci, si: integer;
      begin

   -- convertir tipos bit a enteros
   if a = '0' then. ai := 0;    -- convertir a a entero
            else ai := 1;
   end if;
   if b = '0' then bi := 0;  -- convertir b a entero
            else bi := 1;
   end if;
   if cin = '0' then ci := 0;-- convertir cin a entero
                else ci := 1;
   end if;

   -- calcular la suma entera de las entradas
   si := ai + bi + ci;

   -- convertir el resultado a bits de suma y acarreo
   case si is
      when 0 => s <= '0'; cout <= '0';
      when 1 => s <= '1'; cout <- '0';
      when 2 => s <= '0'; cout <= '1';
      when 3 => s <= '1'; cout <s ,1.,;
   end switch;
end process ;
end full_adder ;
```

Figura 2.38 Modelo de comportamiento VHDL de un circuito sumador completo.

c_{in}, y dos puertos de salida, s y c_{out}, todos declarados como de tipo de datos "bit". La primera línea de la declaración de entidad define el nombre del modelo como "full_adder" (sumador completo).

Si la entidad describe el circuito desde el punto de vista del mundo exterior, la *arquitectura* define la función implantada dentro del circuito. Podemos modelar e implantar un circuito dado de diversas maneras, y por ello VHDL permite definir varias arquitecturas para una entidad dada. El diseñador no tiene que conocer los detalles de la implantación de un circuito para poder usarlo en un diseño de alto nivel; sólo necesita las definiciones de las señales que entran y salen del circuito, así como sus diversos parámetros de tiempos y carga, para poder trabajar. Así, la descripción de entidad proporciona toda la información que necesitaría un usuario del módulo. En la arquitectura del sumador completo de la figura 2.38, la primera línea indica que esta implantación específica de

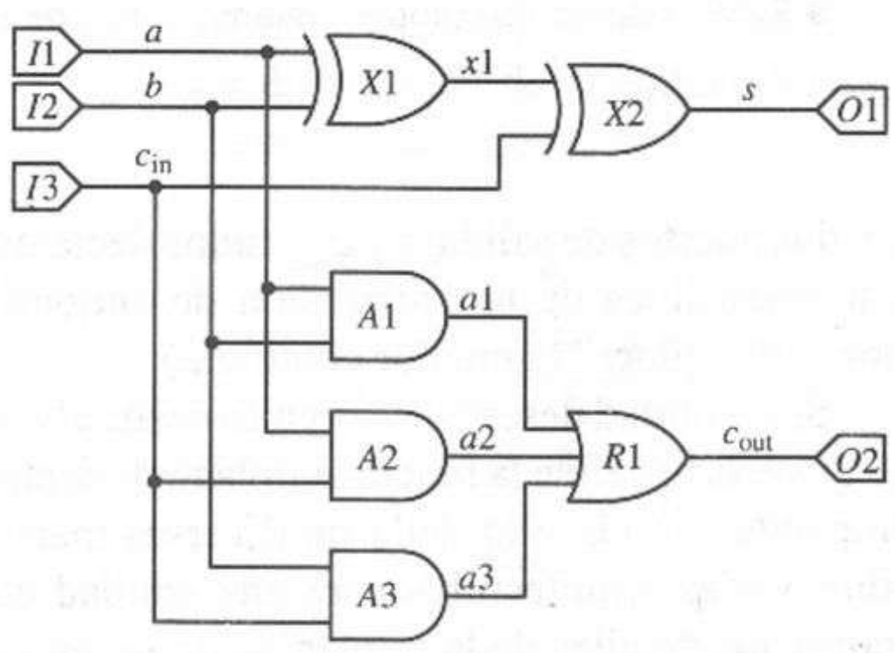
"full_adder" se llama "behavior" (comportamiento). Aquí describimos la función de suma mediante un "process" (proceso) en el que las tres entradas se convierten en valores enteros y se suman para obtener una suma. El resultado se convierte en dos bits distintos, un bit de suma de 0 o 1 y un acarreo de salida si la suma es igual a 2. Las frases que inician con dos guiones son comentarios y no forman parte del modelo. El lector puede consultar [11,12] para obtener mayor información acerca del modelado con VHDL.

Un *modelo estructural* es sólo una interconexión de componentes, es decir, una "estructura", sin un comportamiento específico explícito. Deducimos el comportamiento analizando los modelos de comportamiento de los componentes individuales y su interconexión. El mecanismo más común para representar modelos estructurales de circuitos digitales es el diagrama lógico o *esquemático*. También se utilizan con frecuencia las representaciones textuales de los diagramas esquemáticos, llamadas *listas de red*, al igual que las descripciones HDL de las estructuras de circuito. Las figuras 2.39a y b presentan un modelo estructural del circuito sumador completo antes definido, en formatos de esquema y de lista de red, respectivamente.

El formato de lista de red es particularmente interesante, ya que la mayor parte de los sistemas de CAD necesitan una lista de este tipo para simular la operación de un circuito. Si utilizamos diagramas esquemáticos, por lo general éstos se traducen a la forma de lista de red antes de su simulación. Definimos una *red* como un alambre o una línea de señal lógica cuyo valor se puede controlar o examinar durante la operación del circuito. En una lista de red, definimos cada elemento de circuito como sigue:

nombre_compuerta tipo_compuerta salida entrada1 entrada2 ... entradaN

donde *nombre_compuerta* es un nombre simbólico asignado a esta compuerta en particular, *tipo_compuerta* es el tipo de compuerta lógica (AND2, OR3, etcétera), *salida* es el nombre de una red conectada a la salida de la compuerta y *entrada1* a *entradaN* son los nombres de las redes conectadas a las entradas de



(a)

I1	IN	*a*			
I2	IN	*b*			
I3	IN	c_{in}			
X1	XOR2	*x1*	*a*	*b*	
X2	XOR2	*s*	*x1*	c_{in}	
A1	AND2	*a1*	*a*	*b*	
A2	AND2	*a2*	*a*	c_{in}	
A3	AND2	*a3*	*b*	c_{in}	
R1	OR3	c_{out}	*a1*	*a2*	*a3*
O1	OUT	*s*			
O2	OUT	c_{out}			

(b)

Figura 2.39 Modelos estructurales de un circuito sumador completo. **(a)** Diagrama esquemático. **(b)** Lista de red.

la compuerta. En este circuito y en los ejemplos siguientes, el tipo de compuerta AND2 es una compuerta AND de dos entradas, XOR2 es una compuerta OR exclusiva de dos entradas, y OR3 es una compuerta OR de tres entradas.

La figura 2.39b presenta la lista de red extraída del diagrama esquemático de la figura 2.39a. En este ejemplo tenemos seis compuertas lógicas primitivas y nueve redes. Designamos las redes a, b y c_{in} como entradas externas, mientras que s y c_{out} son salidas externas. Las redes $a1$, $a2$, $a3$ y $x1$ son todas internas al circuito. Observe que los tipos de compuertas IN y OUT no corresponden realmente a compuertas lógicas, sino que designan conexiones externas.

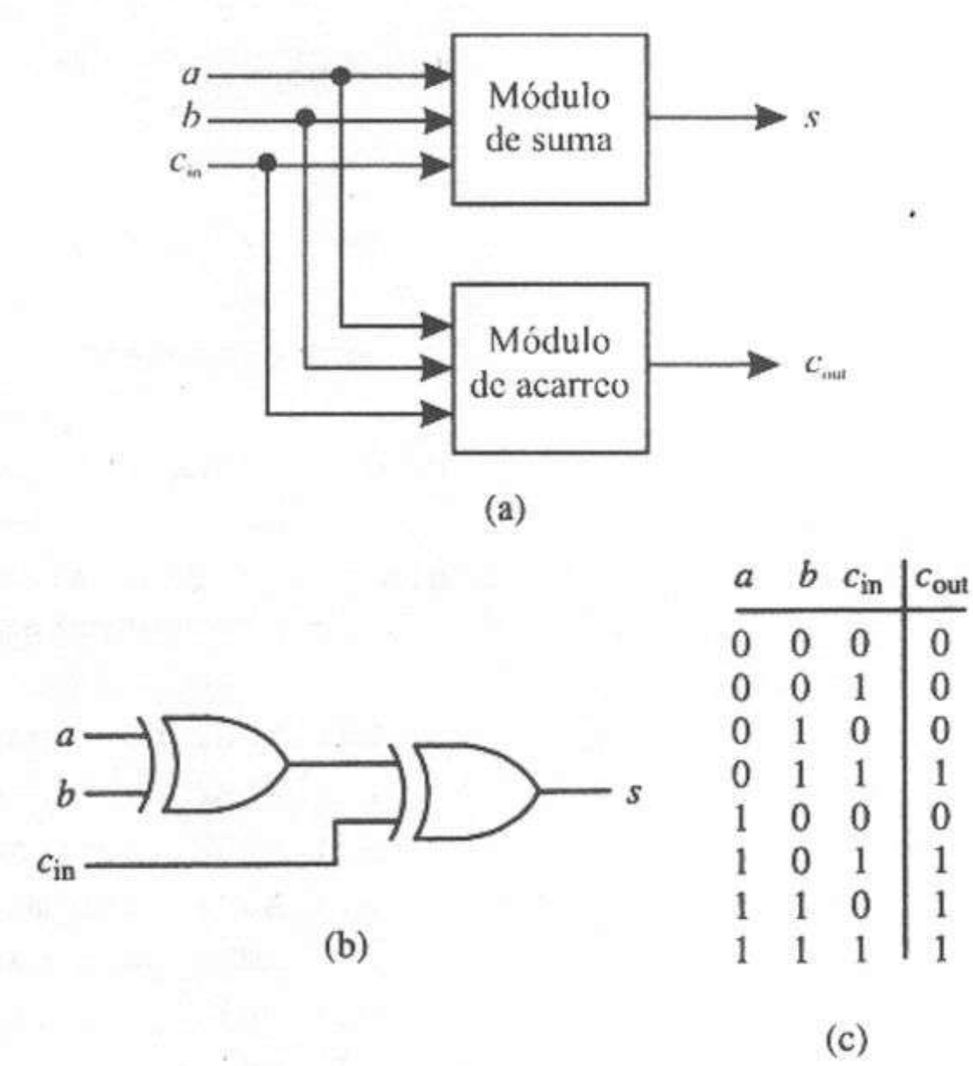
La figura 2.40 muestra una descripción estructural VHDL del esquema del sumador completo de la figura 2.39a. Observe que las descripciones VHDL tanto del modelo estructural como de comportamiento utilizan la misma

```
entity full_adder is
     port(a, b, cin: in bit;
          s, cout: out bit)
end full_adder;

architecture structure of full-adder is
component XOR2          -- declara la compuerta XOR
     port(z: out bit;
          x, y: in bit);
end component;

component AND2          --  declara la compuerta AND
     port(z out bat
          x, y: in bit);
end component;

component OR3           --  declara la compuerta OR
     port(z: out bit,
          w, x, y: in bit);
     end component;

signal xl,al,a2,a3: bit;      --  alambres de señal interna

begin  -- define el esquema conectando los puertos de los
       -- componentes a alambres de señal

     X1: XOR2 port map (xl, a, b);
     X2: XOR2 port map.(s, xl,cin);
     A1: AND2. port map (al, a, b) ;
     A2: AND2 port map (a2; a, cin);
     A3: AND2 port map (a3, b, cin);
     01: OR3     port map (cout, al, a2, a3);
end full_adder ;
```

Figura 2.40 Modelo estructural VHDL de un circuito sumador completo.

definición de entidad. Esto se debe a que la interfaz con el mundo exterior es independiente de la implantación interna de la función. La arquitectura VHDL del modelo estructural define la implantación como una sencilla interconexión de los componentes XOR2, AND2 y OR3. Declaramos cada componente por utilizar mediante una proposición "component" que muestra el nombre de la entidad y del puerto de componente. Al compilar el modelo, obtenemos los modelos de estos componentes de una biblioteca de componentes. En el cuerpo de la arquitectura, las instancias de cada componente por utilizar aparecen en una proposición aparte; cada instancia tiene un nombre único de compuerta. Definimos la estructura del circuito asociando señales (alambres) a los puertos de entrada o salida de los componentes. Por ejemplo, la señal $x1$ de la arquitectura del sumador completo de la figura 2.40 está conectada al puerto de salida z de la compuerta $X1$ de tipo XOR2 y a la entrada x de la compuerta $X2$ de tipo XOR2. De manera similar, las entradas externas a y b están conectadas a los puertos de entrada x y y de la compuerta $X1$ de tipo XOR2. El lector puede consultar [11,12] si desea mayor información sobre la creación de modelos estructurales con VHDL.

Los modelos de circuitos digitales no tienen que ser exclusivamente de comportamiento o estructurales. La mayor parte de los diseños de gran tamaño se desarrollan de forma modular, comenzando con un modelo de comportamiento dividido en módulos. A continuación, se diseñan los circuitos digitales para cada módulo, de manera independiente, hasta realizar todo el circuito. Durante este proceso, algunos de los módulos podrían tener modelos de comportamiento, y otros, modelos estructurales. Los modelos de circuitos que contienen componentes de comportamiento y estructurales son modelos de *modo mixto*. Una vez desarrollado el circuito lógico para cada módulo, se inserta en el modelo



a	b	c_{in}	c_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(c)

Figura 2.41 Modelo de modo mixto para el circuito de sumador completo. **(a)** Diagrama de bloques del sumador completo. **(b)** Circuito para la función suma. **(c)** Tabla de verdad para la función acarreo.

```
entity full_adder is
   port(a; b; cin: in bit;
         a, cout: out bit)
end full_adder;

architecture structure of full_adder is
component XOR2      — declara la compuerta XOR
   port(z: out bit;
         x; y... in bit)

signal x1,a1,a2,a3: bit;  — alambres internos de señales

begin
   — realiza el bit de suma con la estructura de compuerta XOR
   X1: XOR2 port map (x1, a, b);
   X2: XOR2 port map (s, x1; cin);

   — describe el comportamiento del acarreo
   process (a, b, cin)
   begin    — el acarreo de salida es 1 si al menos dos
              entradas son 1
     if a = '1' and b = '1' then
        cout <= '1';
     elsif a = '1' and cin = '1' then
        cout <= '1';
     elsif b = '1' and cin = '1' then
        cout <= '1';
     else
        cout <= '0';
     end if;
   end process;
end full-adder;
```

Figura 2.42 Modelo de modo mixto VHDL para el circuito de sumador completo.

general en vez del comportamiento que realiza, y se verifica todo el modelo para garantizar que el comportamiento global sea el correcto. Esto permite probar cada circuito individual dentro del contexto del diseño global sin esperar a que se desarrolle todo el circuito lógico.

La figura 2.41a muestra un modelo de modo mixto para el sumador completo, dividido en módulos independientes para calcular los bits de suma y de acarreo. En la figura 2.41b se realiza la suma mediante una estructura de dos compuertas XOR. Modelamos la salida de acarreo, como lo muestra la figura 2.41c, mediante su tabla de verdad. El siguiente paso en el diseño sería diseñar el circuito lógico para la salida de acarreo a partir de su tabla de verdad.

La figura 2.42 presenta el mismo modelo de modo mixto para el sumador completo en VHDL. Al igual que en la figura 2.41, la suma se realiza mediante una estructura de dos compuertas XOR, mientras que la salida del acarreo se describe sólo por su comportamiento. Cabe señalar que VHDL, al igual que otros HDL, puede representar circuitos y sistemas con cualquier nivel de

abstracción deseado o con cualquier mezcla de niveles, lo que permite al diseñador trabajar dentro de un mismo entorno desde el concepto hasta la realización del circuito lógico.

2.7.3 Síntesis de diseño y herramientas de captura

El proceso de síntesis del diseño

Como ya hemos dicho, la *síntesis lógica* es el proceso de diseño de un circuito digital a partir de cierta especificación inicial de un problema. Como lo muestra la figura 2.43, la síntesis de un circuito lógico implica varias actividades.

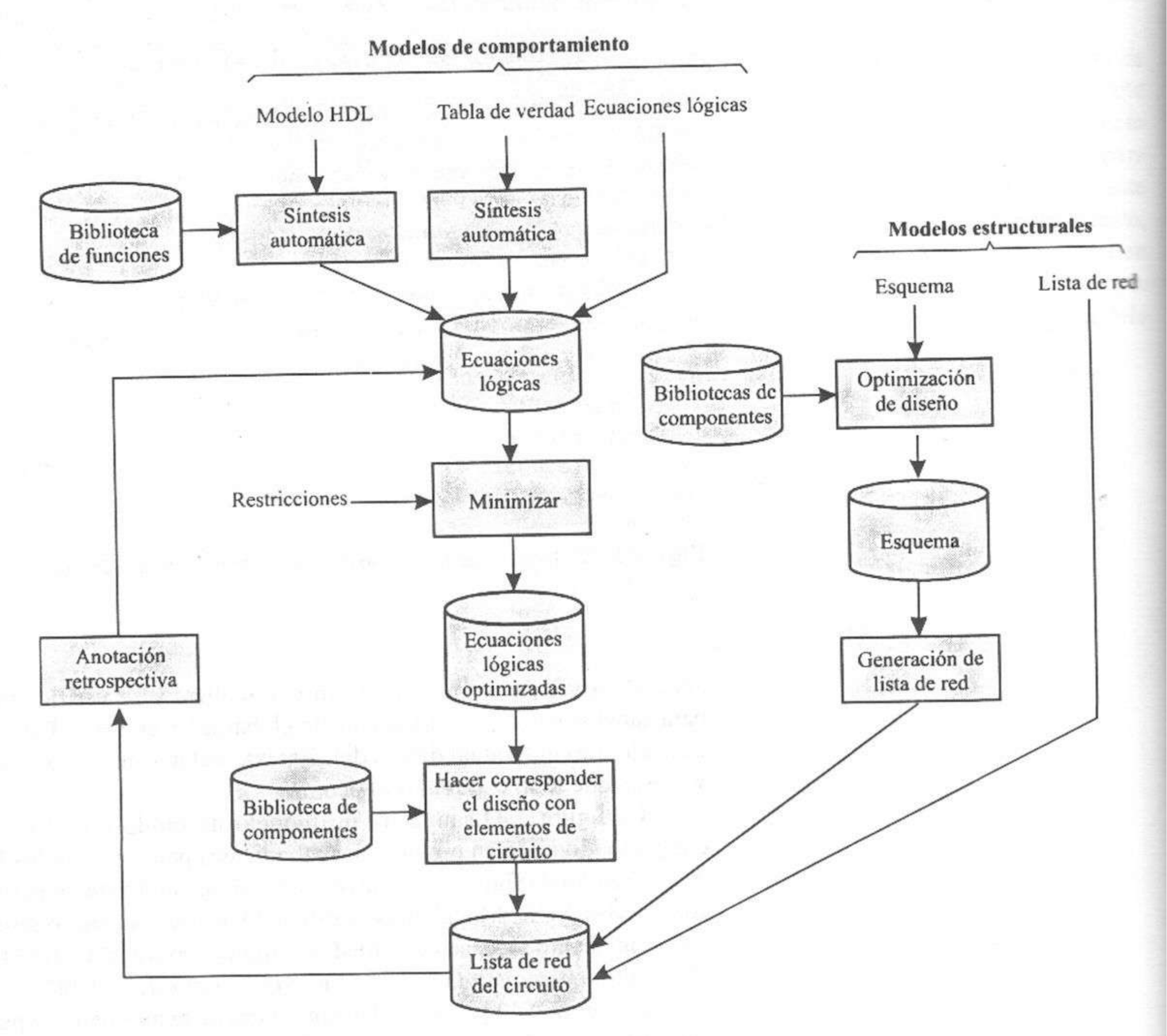


Figura 2.43 Síntesis de un circuito lógico a partir de un modelo de comportamiento o estructural.

Si el diseñador comienza por desarrollar un modelo de comportamiento, debe transformar este modelo en un modelo estructural que pueda realizar con los componentes disponibles. Esta transformación comienza con la deducción de un conjunto de ecuaciones lógicas a partir del modelo. Después, se reducen estas ecuaciones para satisfacer ciertas restricciones de diseño, las cuales pueden incluir limitaciones sobre el costo total, el número o los tipos de compuertas por utilizar, el *fan-in* o *fan-out* de las compuertas, el espacio físico (lo que restringe el número de compuertas o paquetes de circuitos), las características de tiempos (por ejemplo, restricción del diseño a dos niveles, en lugar de tres o más), el consumo de energía, y el tiempo de diseño. Por último, las ecuaciones reducidas se asocian a elementos específicos del circuito. El resultado final es una lista de red de circuitos lógicos que indica los elementos utilizados, así como su interconexión.

Algunos o todos estos pasos de la transformación se pueden realizar en forma automática mediante herramientas de CAD especiales. En una sección posterior de este capítulo analizaremos la minimización del diseño con mayor detalle. El lector debe consultar [14] si desea mayor información acerca de la síntesis automática de los circuitos lógicos a partir de modelos de comportamiento.

Si el diseñador desarrolla un modelo estructural en forma de esquema o de lista de red, en vez de partir de un modelo de comportamiento, los pasos anteriores (deducción y minimización de las ecuaciones lógicas y asociación de éstas a componentes del circuito) deben realizarse en forma manual. En este caso, sólo se guarda el circuito final en la base de datos final del circuito. Sin embargo, muchos sistemas de CAD incluyen una función llamada *anotación en retrospectiva* (*back annotation*), que se muestra en la figura 2.43 y que extrae las ecuaciones lógicas de una lista de red de esquema retroalimentada al proceso de diseño. Esto permite procesar el diseño con las herramientas de minimización y reasociarlas en algunos componentes seleccionados del circuito para realizar mejoras y las correcciones deseadas.

Cada modelo de circuito creado en el proceso de diseño debe capturarse con un formato susceptible de ser guardado y procesado por una computadora digital. Las herramientas de CAD utilizadas en la captura del diseño pueden variar según el tipo de modelo. Los diagramas esquemáticos se crean mediante un programa para *captura de esquemas* o *editor de esquemas*, el cual proporciona un entorno gráfico interactivo en el que el diseñador puede trazar y editar dibujos esquemáticos. Los formatos no esquemáticos de los modelos se crean y guardan en un sistema de CAD como archivos de texto en ASCII y, por tanto, se pueden desarrollar por medio de editores de texto estándar. Algunos sistemas de CAD proporcionan editores de texto especiales adaptados a un HDL, lista de red u otro formato de modelo en particular. Estos editores especiales ayudan a crear el formato del modelo y a verificar errores, y facilitan la creación del modelo con plantillas que se pueden llenar con nombres de elementos y parámetros.

Antes de que una computadora pueda procesar un diseño, éste debe traducirse a un formato intermedio que pueda ser manipulado por las diversas herramientas de CAD. Este formato intermedio sería independiente del método utilizado para desarrollar el modelo. Durante la traducción, por lo general se verifican los errores del modelo. Estos errores incluyen los de conectividad, como las entradas de compuerta no conectadas, varias compuertas que controlan una única línea, y una salida de compuerta que controla demasiadas entradas de compuerta, así como errores de redes sin nombre o parámetros de circuito faltantes, inadecuados o inconsistentes. Éstos se conocen como *errores de regla de diseño*, puesto que violan las reglas básicas de diseño de un circuito digital.

Sin embargo, estas verificaciones no determinan que una función sea correcta desde el punto de vista lógico. Esto debe comprobarse mediante simulación lógica, que analizaremos más adelante.

De manera análoga, se verifican los errores en los diseños representados con un HDL, como VHDL, mientras se traducen a formatos intermedios, de la misma forma en que los programas estándar en FORTRAN o C deben verificarse en búsqueda de errores de sintaxis al momento de su compilación. Los errores detectados durante la traducción incluyen los errores de sintaxis del lenguaje, así como los errores de regla de diseño similares a los ya descritos.

Captura de esquemas

Un programa de captura de esquemas es una herramienta gráfica interactiva con la que un diseñador "dibuja" un diagrama lógico que debe ser procesado por un sistema de CAD. Los pasos básicos del proceso de captura de esquemas son los siguientes:

1. Crear (o "abrir") una hoja de dibujo.
2. Seleccionar los componentes de una biblioteca y colocarlos en la hoja de dibujo.
3. Trazar redes (alambres) para interconectar los componentes.
4. Asignar nombres simbólicos a cada componente y red.
5. Definir o ajustar los parámetros característicos de los componentes.
6. Verificar los errores del esquema.
7. Guardar el esquema en la base de datos.

En la captura de esquema, un solo símbolo de componente puede representar un elemento lógico primitivo o todo un módulo de circuito. Un *componente primitivo* es aquel que no se forma con elementos menores. Representamos cada componente primitivo mediante un símbolo gráfico, que se usará para trazar el esquema, y un modelo subyacente que describe su comportamiento, el cual se utilizará durante la simulación. En el diseño de los circuitos lógicos digitales, los componentes primitivos incluyen por lo general las compuertas lógicas básicas, los conectores de entrada y de salida, los latches y *flip-flops* (definiremos los latches y *flip-flops* en el capítulo 6). Los conectores de entrada y de salida no son en realidad elementos de circuito, pero sirven para identificar las conexiones con las señales externas.

Los componentes están dentro de una o más *bibliotecas* en la base de datos del sistema de CAD, de la cual se recuperan en caso necesario. Por lo general, un sistema de CAD tiene una o más bibliotecas de componentes primitivos estándar, lo que proporciona al diseñador una colección de compuertas lógicas de uso común y elementos similares. En el esquema del sumador completo de la figura 2.39a, elegimos los elementos lógicos AND2 (compuerta AND de dos entradas), XOR2 (compuerta OR exclusivo de dos entradas), OR3 (compuerta OR con tres entradas), IN (conector para una señal de entrada externa) y OUT (conector para una señal de salida externa) de una biblioteca de compuertas lógicas primitivas.

Utilizamos las bibliotecas de circuitos no primitivos para crear diseños modulares jerárquicos, de la forma que describiremos en el capítulo 4. Estas bibliotecas pueden ser proporcionadas por el proveedor del sistema de CAD o bien obtenerse de un tercero, como el fabricante de los circuitos VLSI o tarjetas de circuitos que se diseñarán con este sistema. La mayor parte de los sistemas de

CAD también manejan bibliotecas creadas por el usuario, lo que permite a un diseñador individual desarrollar y guardar diseños de circuitos en una biblioteca especial para uso futuro. En el resto de este capítulo nos centraremos en el modelado de circuitos lógicos combinatorios con compuertas primitivas.

La figura 2.44 ilustra el proceso de captura de esquemas, utilizando como ejemplo el circuito de sumador completo de la figura 2.39a. La figura 2.44a muestra una pantalla típica para captura de esquemas, donde aparece un menú de operaciones a la izquierda del área de dibujo. Realizamos una operación colocando el cursor sobre la opción de menú con un ratón o algún dispositivo apuntador similar, y haciendo clic con un botón del ratón o bien oprimiendo una tecla para iniciar la operación. Para este análisis, nos referiremos a este proceso como hacer clic sobre una opción. La mayor parte de los sistemas también permite introducir las instrucciones desde un teclado.

Abrimos una hoja de dibujo haciendo clic sobre la opción OPEN SHEET del menú. Después seleccionamos componentes de la biblioteca y lo colocamos

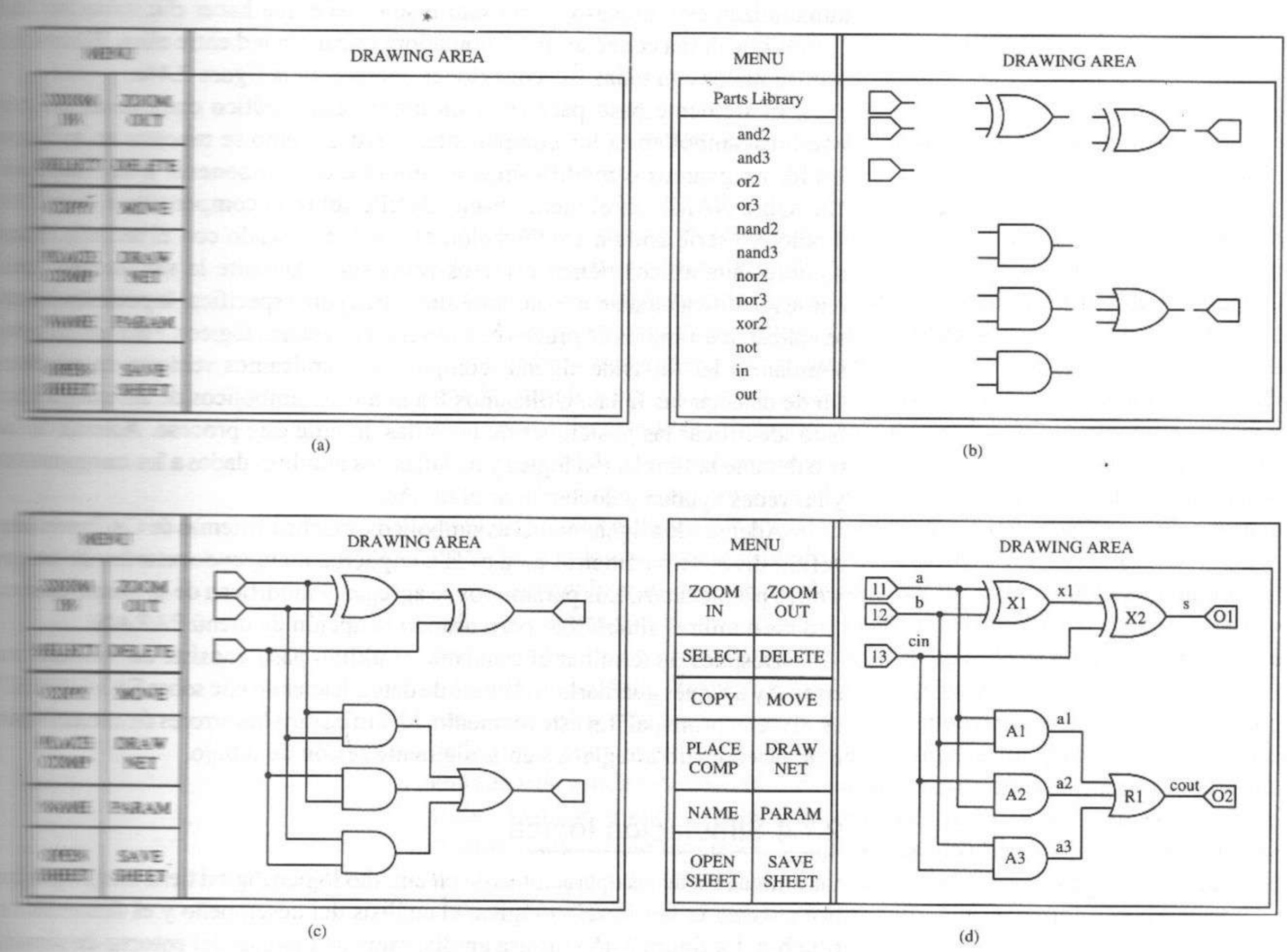


Figura 2.44 El proceso de captura de esquemas. **(a)** Menú principal y área de dibujo. **(b)** Compuertas seleccionadas mediante una biblioteca. **(c)** Redes trazadas para conectar compuertas. **(d)** Nombres agregados a las compuertas y las redes.

sobre la hoja. Al hacer clic sobre PLACE COMP, se despliega un menú de componentes disponibles, como se muestra en la figura 2.44b. Seleccionamos un componente de la biblioteca si hacemos clic sobre él en el menú. Después, el componente seleccionado se coloca en la hoja de dibujo llevando el cursor hasta la posición deseada y haciendo clic de nuevo. Una vez en la hoja, podemos copiar, mover o eliminar un componente, haciendo clic sobre éste para seleccionarlo y haciendo clic después sobre COPY, MOVE o DELETE en el menú principal. Otras propiedades usuales son el cambio de escala y la rotación de los componentes. La figura 2.44b muestra el dibujo del sumador completo después de colocar todas las compuertas en la hoja.

Una vez colocados los componentes en la hoja, éstos deben conectarse mediante redes (alambres). Un extremo de una red se conecta a un componente haciendo clic sobre una de las terminales de entrada o de salida del componente. Después, se traza la red moviendo el cursor; se traza un segmento de la red desde el extremo final inicial hasta el cursor. La red se termina haciendo clic sobre la terminal de la compuerta o red a la que debe conectarse. Las redes con varios segmentos se crean segmento por segmento, llevando el cursor al punto final de cada segmento y haciendo clic sobre él. Algunos sistemas de CAD automatizan este proceso, y el usuario sólo tiene que hacer clic sobre los dos puntos que desea conectar, la computadora trazará la red entre ellos. El circuito semisumador con todas sus conexiones aparece en la figura 2.44c.

El siguiente paso para crear un dibujo esquemático consiste en asignar nombres simbólicos a los componentes y redes, como se muestra en la figura 2.44d. Asignamos o modificamos un nombre de componente o red haciendo clic sobre NAME en el menú, haciendo clic sobre el componente o la red del dibujo, y escribiendo a continuación el nombre deseado con el teclado. Estos nombres simbólicos tienen diversos propósitos. Durante la simulación, cada red se identifica mediante su nombre simbólico para especificar la posición donde se aplicarán estímulos de prueba o se observarán valores lógicos. Para una prueba, simulamos las fallas de algunas compuertas y aplicamos vectores de prueba a fin de detectar las fallas. Utilizamos los nombres simbólicos de las compuertas para identificar las posiciones de las fallas durante este proceso. Además de su uso durante la simulación lógica y de fallas, los nombres dados a los componentes y las redes ayudan a documentar el diseño.

Además de asignar nombres simbólicos, muchos sistemas de CAD permiten definir diversos parámetros para cada compuerta, incluyendo retardos de tiempo y otras propiedades. Los parámetros se agregan y modifican de la misma manera que los nombres simbólicos pero usando la opción de menú PARAM.

Después de terminar el esquema, el último paso consiste en verificar sus errores y después guardarlo en la base de datos, haciendo clic sobre SAVE SHEET en el menú principal. En este momento, identificamos los errores de conectividad o de diseño para arreglarlos en la siguiente sesión de dibujo.

2.7.4 Simulación lógica

La simulación de las operaciones de un circuito lógico digital tiene tres propósitos principales: la verificación lógica, el análisis del desempeño y el desarrollo de pruebas. La figura 2.45 muestra un diagrama de bloques del entorno de simulación. Por lo general, el modelo del circuito se alimenta al simulador como una lista de red plana. El *aplanamiento* es el proceso de reemplazo de todos los módulos del circuito no primitivos por las redes de elementos lógicos primitivos

que representan. Para procesar una lista de red, el simulador obtiene de una biblioteca de componentes primitivos los modelos de simulación de las compuertas lógicas dadas en la lista de red.

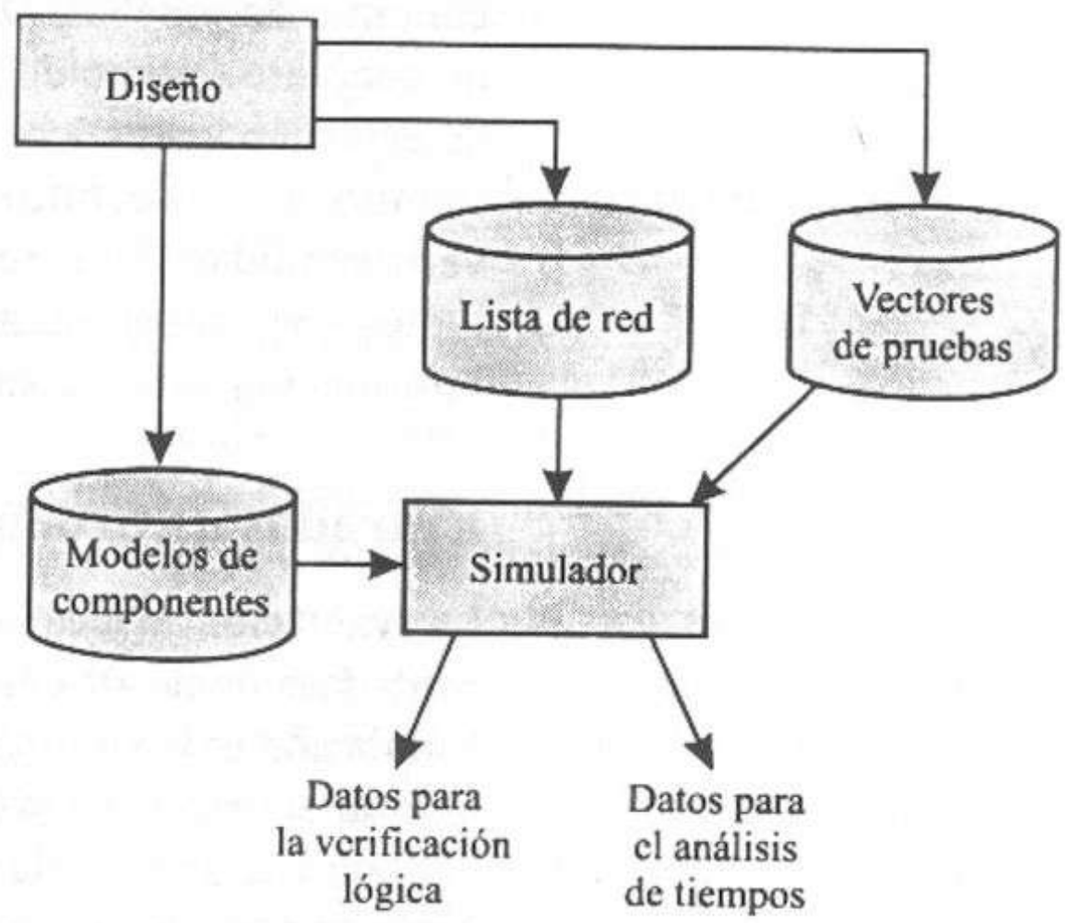


Figura 2.45 Entorno de simulación lógica digital.

Verificamos un diseño de circuito lógico aplicando vectores de prueba para estimular las entradas del circuito. Un *vector de prueba* es una lista ordenada de unos y ceros, cada uno de los cuales corresponde a un valor de estímulo por aplicar a una entrada de circuito dada. Después se capturan las respuestas de salida de las compuertas seleccionadas y se analiza si son correctas, comparando con la tabla de verdad, las ecuaciones lógicas u otras especificaciones utilizadas para desarrollar el modelo. En las primeras etapas del ciclo de diseño, lo principal es que el modelo sea correcto desde el punto de vista lógico. Por tanto, se usan modelos de componentes simplificados o "ideales", los cuales no tienen retardos en su respuesta a estímulos de entrada. Esto separa la información relativa tiempos de la función lógica, a fin de simplificar los análisis preliminares.

Para analizar el desempeño de un circuito, cada modelo de componente debe aproximar con alta fidelidad las características físicas de su dispositivo correspondiente. En particular, el tiempo de respuesta de un dispositivo a cierto estímulo de entrada, llamado *retardo por propagación*, debe modelarse con precisión. Al contar con modelos precisos, podemos analizar los resultados de la simulación y predecir los retardos de propagación globales entre *pines* de entrada y de salida seleccionados de un circuito (*retardo de pin a pin*). Además, podemos detectar otros problemas potenciales de tiempos, como los picos y los riesgos que describiremos en el capítulo 3. Cuando el diseñador enfrenta opciones de diseño, la simulación proporciona un método para evaluar los efectos de cada opción sobre el desempeño del circuito, lo que permite hacer elecciones óptimas.

Una vez fabricado un circuito, se debe probar para determinar si contiene componentes o líneas de señal defectuosos. Para cada falla potencial, se aplican los vectores de prueba que producirán salidas del circuito defectuoso diferentes

de las que surgen de los circuitos sin fallas, lo que permite al verificador determinar si la falla está presente. La *simulación de fallas* es el proceso de simulación de la presencia de varias fallas (*inyección de fallas*), para determinar si son detectables mediante un conjunto dado de vectores de prueba. Por lo general, los resultados indican el porcentaje de fallas detectables mediante un conjunto de pruebas. De ese modo, la simulación de fallas ayuda a desarrollar un conjunto óptimo de prueba para un circuito lógico, es decir, una prueba que detecte un porcentaje aceptable de fallas potenciales en una cantidad mínima de tiempo. En el capítulo 12 analizaremos las pruebas y el diseño para la verificabilidad. En el resto de este capítulo sólo consideraremos la verificación lógica y el análisis de desempeño. El lector debe consultar [15] si desea mayor información sobre la simulación de fallas y la verificación.

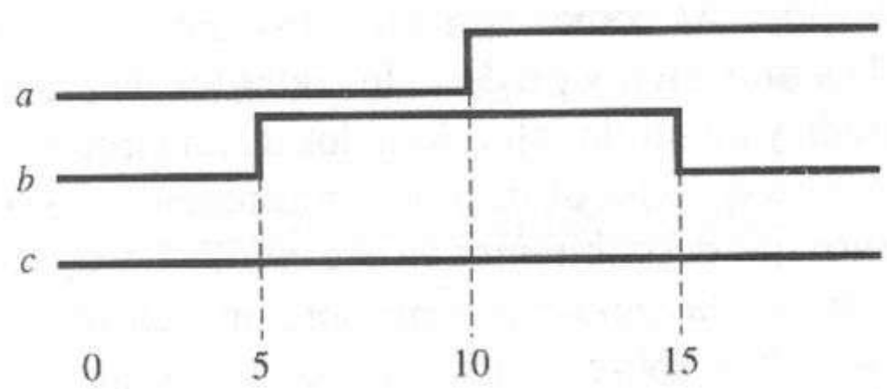
Entradas de prueba para simulación

La evaluación adecuada de un diseño requiere un conjunto de entradas de prueba cuidadosamente diseñado, llamado *conjunto de prueba*. Si el objetivo de la simulación es la verificación lógica de un circuito lógico combinatorio, podemos ignorar el tiempo en la especificación de las entradas de prueba. En este caso, el conjunto de prueba incluye una lista de vectores de entrada que serán aplicados uno tras otro. Al aplicar cada vector de entrada, calculamos y registramos las salidas del circuito; después, aplicamos la siguiente entrada, y así sucesivamente. La figura 2.46 muestra un conjunto de prueba para el circuito del sumador completo de la figura 2.39a. Ésta es una prueba *exhaustiva*; es decir, usamos todas las combinaciones posibles de entradas, lo cual permite verificar toda la tabla de verdad. En general, necesitamos 2^n vectores para verificar exhaustivamente un circuito lógico combinatorio con n entradas, lo que no sería práctico para circuitos con muchas entradas. En estos casos, diseñamos los conjuntos de prueba de modo que permitan verificar las operaciones de los circuitos más comunes y críticas, dando al diseñador cierto grado de seguridad de que el diseño no tiene errores. La cantidad y naturaleza de los vectores de prueba utilizados determinan la magnitud del grado de seguridad.

a	b	c_{in}
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Figura 2.46 Conjunto funcional de prueba para el circuito del sumador completo.

Si hay que estudiar los tiempos del circuito, entonces debemos aplicar las entradas de prueba en momentos específicos, de modo que el circuito tenga tiempo de responder a cada vector. Por tanto, debemos especificar el instante de cada cambio de entrada con cada vector en el conjunto de prueba. La figura 2.47a muestra un conjunto de formas de onda que se aplicará a las entradas a, b y c de un circuito. Las figuras 2.47a y c ilustran dos formatos diferentes de uso frecuente para especificar las formas de onda de las entradas de prueba. En la



(a)

Instante	a	b	c_{in}
0	0	0	0
5	0	1	0
10	1	1	0
15	1	0	0

(b)

a = 0:0, 10:1;
b = 0:0, 5:1,15:0;
c_{in} = 0:0;

(c)

Figura 2.47 Conjunto de prueba para el sumador completo, que especifica las formas de onda de entrada. **(a)** Formas de onda de entradas para prueba. **(b)** Formato de tabla. **(c)** Formato de forma de onda.

figura 2.47b, enumeramos el conjunto de prueba en forma de tabla, organizado con un vector de prueba por línea. El primer número en cada línea es el instante de la simulación en que se aplicará el vector. Esta misma prueba tiene otro formato en la figura 2.47c. En este caso, cada línea incluye un nombre de entrada y la especificación de la forma de onda de una señal que se aplicará a esa entrada. Una *forma de onda* se especifica mediante una lista de pares {tiempo:valor}, cada uno de los cuales indica el instante en que variará la forma de onda y el valor al que debe cambiar la señal. El lector debe comprobar que los conjuntos de prueba de las figuras 2.47b y c describen la misma verificación.

Simulación controlada por eventos

Algunos programas de simulación lógica calculan la salida de cada compuerta lógica durante cada intervalo de tiempo simulado. Sin embargo, la mayor parte de los simuladores de circuitos lógicos digitales son *controlados por eventos*, donde un *evento* se define como un cambio en el valor de una señal en un instante dado. Por ejemplo, consideremos la compuerta AND de la figura 2.48a y el diagrama de tiempos de la figura 2.48b. En el instante T_0, las entradas de la compuerta AND son $a = b = 1$, como se ilustra en el diagrama de tiempos. En el instante T_1 ocurre el evento (T_1, a, 0), es decir, en el instante T_1 la señal a cambia de su valor presente a 0. Como resultado de este evento sobre la entrada a, la salida de la compuerta AND, c, cambiará de 1 a 0 en el instante $T_1 + \Delta t$, donde Δt es el retardo por propagación en la compuerta AND. Por tanto, el evento ($T_1 + \Delta t$, c, 0) es consecuencia del evento (T_1, a, 0). Supongamos ahora que ocurre el evento (T_2, b, 0), es decir, la entrada b cambia de 1 a 0 en el instante T_2. Como la salida de la compuerta AND, c, ya es igual a 0, no es afectada por el nuevo valor de b, y no se disparan eventos adicionales. De esta manera, cada evento de entrada se propaga a través del circuito hasta alcanzar una salida del circuito o hasta que ya no hay más cambios de señales; es decir, los eventos "controlan" la simulación.

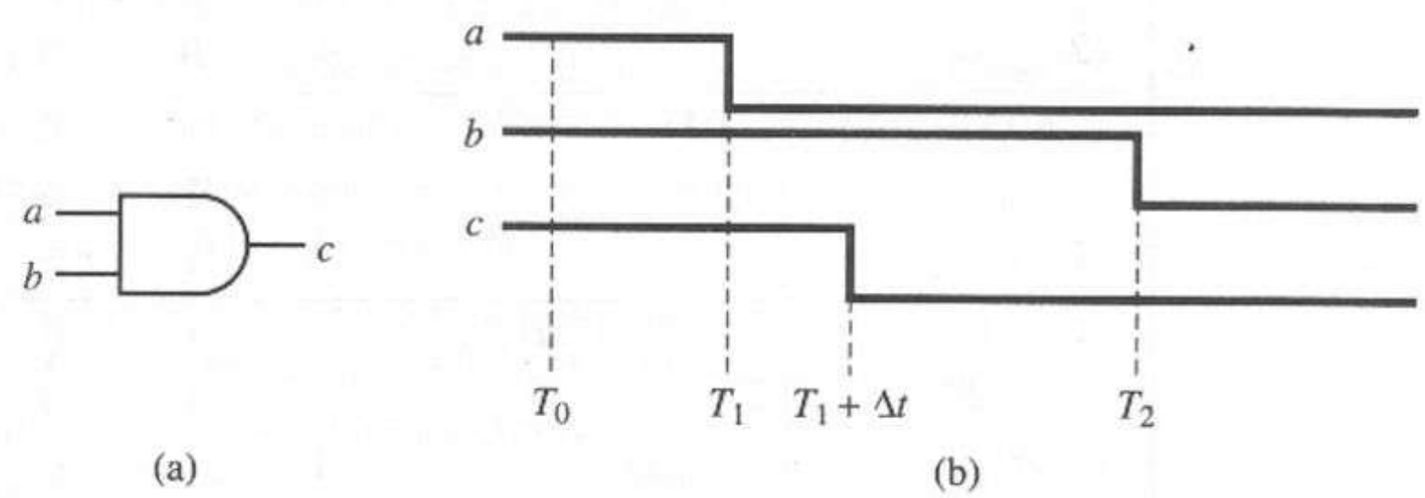


Figura 2.48 Ejemplo de simulación controlada por eventos. **(a)** Modelo de compuerta AND. **(b)** Diagrama de tiempos.

Un simulador controlado por eventos se organiza en torno a una *cola* o *lista de eventos*, donde los eventos se guardan según el orden de su tiempo programado de ocurrencia. En cada paso de simulación, se elimina el primer evento de la cola; es decir, el siguiente evento programado para ocurrir. Se hace que el evento ocurra cambiando el valor de la red indicada. Si esta red es una entrada de una o más compuertas, entonces se calcula de nuevo la salida de cada compuerta afectada. Para cada salida calculada que difiera de su salida actual,

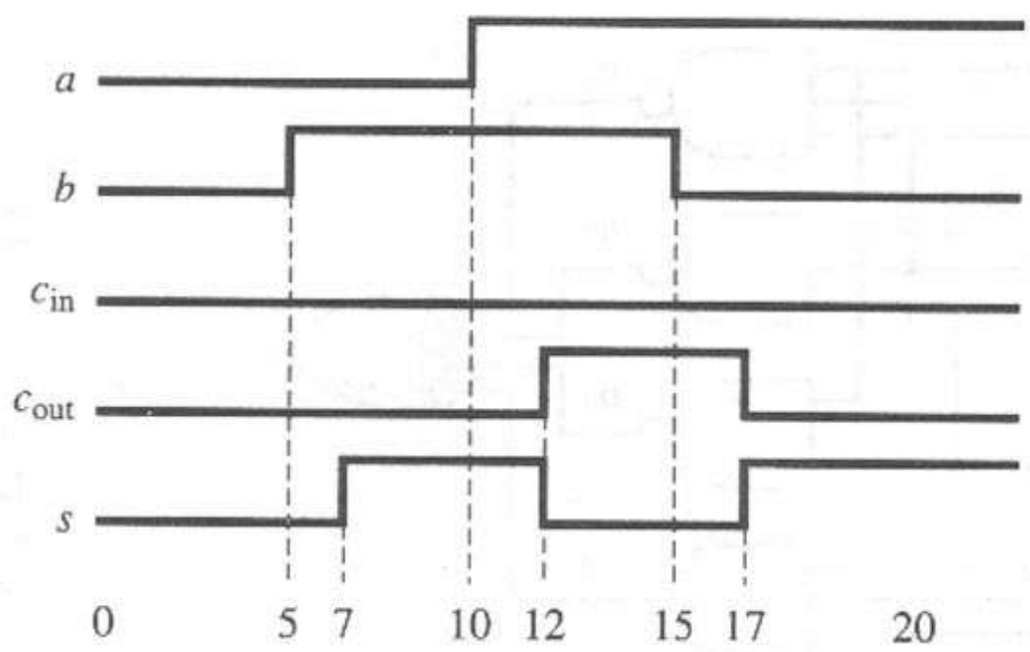
se crea un nuevo evento para la red controlada por esa salida y se coloca en la cola de eventos. El tiempo en que está programado este evento se calcula como el tiempo actual más el retardo de la compuerta. De esta manera, continúa la simulación hasta que se vacía la cola de eventos o hasta que transcurre un intervalo de tiempo dado.

La simulación se inicia al convertir el conjunto de prueba de entrada en un conjunto de eventos e insertar éstos en la cola de eventos en sus tiempos programados. Durante la simulación, se mantiene un registro de todos los eventos del que se pueden generar y examinar resultados de la simulación, ya sea en forma de tabla o de diagrama de tiempos. La figura 2.49 ilustra la serie de eventos que se presentarían durante una simulación del circuito del sumador completo de la figura 2.39a con el conjunto de prueba de la figura 2.47a. En este ejemplo, hemos supuesto que cada compuerta tiene un retardo de una unidad de tiempo. En la figura, el valor x representa el valor inicial desconocido de una señal lógica.

Podemos exhibir los resultados de la simulación, mostrando sólo las entradas y salidas externas, como formas de onda (figura 2.50a) o en un formato de tabla (figuras 2.50b y c) que contiene muestras de las formas de onda tomadas en los instantes indicados. Por lo general, el diseñador especifica las señales por

Evento	**Eventos resultantes**	a	b	c_{in}	$a1$	$a2$	$a3$	$x1$	s	c_{out}
Estado inicial		x	x	x	x	x	x	x	x	x
$(0,a,0)$	$(1,a1,0),(1,a2,0)$	0	x	x	x	x	x	x	x	x
$(0,b,0)$	$(1,x1,0),(1,a3,0)$	0	0	x	x	x	x	x	x	x
$(0,c_{in},0)$	$(1,a1,0),(1,a2,0)$	0	0	0	x	x	x	x	x	x
$(1,a1,0)$		0	0	0	0	x	x	x	x	x
$(1,a2,0)$		0	0	0	0	0	x	x	x	x
$(1,x1,0)$	$(2,s,0)$	0	0	0	0	0	x	0	x	x
$(1,a3,0)$	$(2,c_{out},0)$	0	0	0	0	0	0	0	x	x
$(2,s,0)$		0	0	0	0	0	0	0	0	x
$(2,c_{out},0)$		0	0	0	0	0	0	0	0	0
$(5,b,1)$	$(6,x1,1)$	0	1	0	0	0	0	0	0	0
$(6,x1,1)$	$(7,s,1)$	0	1	0	0	0	0	1	0	0
$(7,s,1)$		0	1	0	0	0	0	1	1	0
$(10,a,1)$	$(11,x1,0),(11,a1,1)$	1	1	0	0	0	0	1	1	0
$(11,x1,0)$	$(12,s,0)$	1	1	0	0	0	0	0	1	0
$(11,a1,1)$	$(12,c_{out},1)$	1	1	0	1	0	0	0	1	0
$(12,s,0)$		1	1	0	1	0	0	0	0	0
$(12,c_{out},1)$		1	1	0	1	0	0	0	0	1
$(15,b,0)$	$(16,x1,1),(16,a1,0)$	1	0	0	1	0	0	0	0	1
$(16,x1,1)$	$(17,s,1)$	1	0	0	1	0	0	1	0	1
$(16,a1,0)$	$(17,c_{out},1)$	1	0	0	0	0	0	1	0	1
$(17,s,1)$		1	0	0	0	0	0	1	1	1
$(17,c_{out},1)$		1	0	0	0	0	0	1	0	1

Figura 2.49 Serie de eventos en la simulación del sumador completo.



(a)

Instante	a	b	c_{in}	c_{out}	s
0	0	0	0	X	X
2	0	0	0	0	0
4	0	0	0	0	0
6	0	1	0	0	0
8	0	1	0	0	1
10	1	1	0	0	1
12	1	1	0	1	0
14	1	1	0	1	0
16	1	0	0	1	0
18	1	0	0	0	1
20	1	0	0	0	1

(b)

Instante	a	b	c_{in}	c_{out}	s
0	0	0	0	X	X
2	0	0	0	0	0
5	0	1	0	0	0
7	0	1	0	0	1
10	1	1	0	0	1
12	1	1	0	1	0
15	1	0	0	1	0
17	1	0	0	0	1

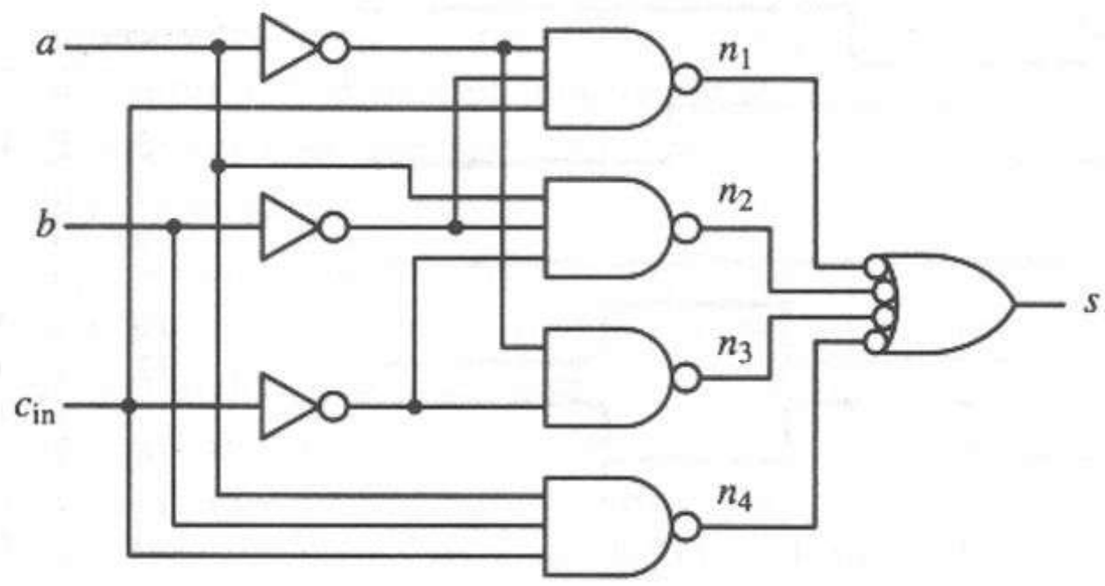
(c)

Figura 2.50 Resultados de la simulación del sumador completo. **(a)** Formato de forma de onda. **(b)** Muestras a intervalos fijos. **(c)** Muestras en cada evento.

exhibir y con frecuencia puede generar solicitudes de varias exhibiciones para una simulación dada, con información distinta cada vez. La tabla de la figura 2.50b enumera los valores de las señales seleccionadas consideradas cada dos unidades de tiempo, mientras que la figura 2.50c enumera los valores sólo cuando ocurren eventos. La última forma facilita al diseñador la identificación de los eventos significativos durante la simulación. Sin embargo, si el número de eventos es muy grande, es preferible tomar muestras a intervalos fijos, con el intervalo de muestra seleccionado para limitar el número total de muestras.

A fin de ilustrar el uso de la simulación para la verificación de una función lógica, depuraremos el circuito sumador completo de la figura 2.51a, que contiene un error. Suponemos que cada compuerta e inversor tiene un retardo de una unidad de tiempo. Al aplicar el conjunto de prueba de la figura 2.47 obtenemos los resultados de simulación que se muestran en la figura 2.51b. Si comparamos los resultados con la tabla de verdad para el sumador completo de la figura 2.37b, vemos que la salida s en el instante 3 es incorrecta para el vector de entrada $abc_{in} = 000$ (la salida s debe ser 0, no 1), aunque es correcta para los demás vectores de entrada. Para aislar la fuente del error, capturamos más información agregando las señales n_1, n_2, n_3 y n_4 a las salidas, como se muestra en la figura 2.51c. Al examinar estas salidas, vemos que la red $n_3 = 0$ en el instante 2 para el vector de entrada $abc_{in} = 000$, pero el valor correcto debería ser $n_3 = 1$. Esto nos guía hacia la entrada de esa compuerta NAND, donde vemos que falta una conexión a esta compuerta desde la entrada b; es decir, la compuerta realiza la expresión $\overline{a}\overline{c}$ en vez de $\overline{a}b\overline{c}$. Por tanto, hemos identificado el error y ahora podemos corregirlo, para volver a simular y verificar su operación adecuada.

También podemos utilizar la simulación lógica para identificar los riesgos potenciales de un circuito lógico. Por ejemplo, sospechamos que el circuito lógico de la figura 2.52a tiene un riesgo estático. Si todas las entradas son inicialmente iguales a 1 y aplicamos el vector de prueba $(a, b, c) = 011$ en el instante t_1, la simulación produce las formas de onda de la figura 2.52b, suponiendo que cada compuerta tiene un retardo Δt. Se espera que la salida g sea constante, puesto que $g = 1$ para ambos vectores de entrada. Sin embargo, queda claro por la forma de onda de salida que existe un error en la salida en el instante t_3. Al analizar las entradas de la compuerta OR, vemos que el error se deba a que e y



(a)

Instante	a	b	c_{in}	s
0	0	0	0	X
3	0	0	0	1
5	0	1	0	1
10	1	1	0	1
13	1	1	0	0
15	1	0	0	0
18	1	0	0	1

(b)

Instante	a	b	c_{in}	n_1	n_2	n_3	n_4	s
0	0	0	0	X	X	X	X	X
1	0	0	0	1	1	X	1	X
2	0	0	0	1	1	0	1	X
3	0	0	0	1	1	0	1	1
5	0	1	0	1	1	0	1	1
10	1	1	0	1	1	0	1	1
12	1	1	0	1	1	1	1	1
13	1	1	0	1	1	1	1	0
15	1	0	0	1	1	1	1	0
17	1	0	0	1	0	1	1	0
18	1	0	0	1	0	1	1	1

(c)

Figura 2.51 Depuración de un sumador completo con un error. **(a)** Circuito sumador completo erróneo. **(b)** Salida de simulación que muestra un error en s en el instante 3. **(c)** Resultados ampliados de la simulación, aislando el error en n_3.

f son 0 momentáneamente entre los instantes t_2 y t_3, hasta que f cambia a 1, lo cual hace que g sea otra vez igual a 1. Al haber identificado el riesgo, podemos aplicar los procedimientos que veremos en el capítulo 3 para eliminarlo.

No todos los simuladores son controlados por eventos. En algunos casos, se transforma el modelo del circuito en un programa de computadora que después se compila y ejecuta en la computadora anfitriona, como cualquier otro programa. En vez de procesar eventos, se vuelve a calcular la salida de cada compuerta durante cada intervalo de tiempo. Durante la transformación del modelo, las compuertas se ordenan de modo que las salidas de cada una dependan sólo de las entradas externas y de las salidas de compuertas ya calculadas durante el intervalo de tiempo en curso. La principal ventaja de contar con un modelo compilado y un orden cuidadoso de las compuertas es que la velocidad de ejecución de una simulación compilada es por lo general más rápida que la de cualquier simulación controlada por eventos, en términos del número de simulaciones de compuertas ejecutadas por unidad de tiempo. Sin embargo, como los simuladores compilados vuelven a calcular cada salida de compuerta durante cada intervalo de tiempo de simulación, sea que las entradas evaluación hayan cambiado o no, gran parte del tiempo se dedica a la evaluación de compuertas *inertes*, es decir, compuertas cuyas entradas no han cambiado y cuyas salidas,

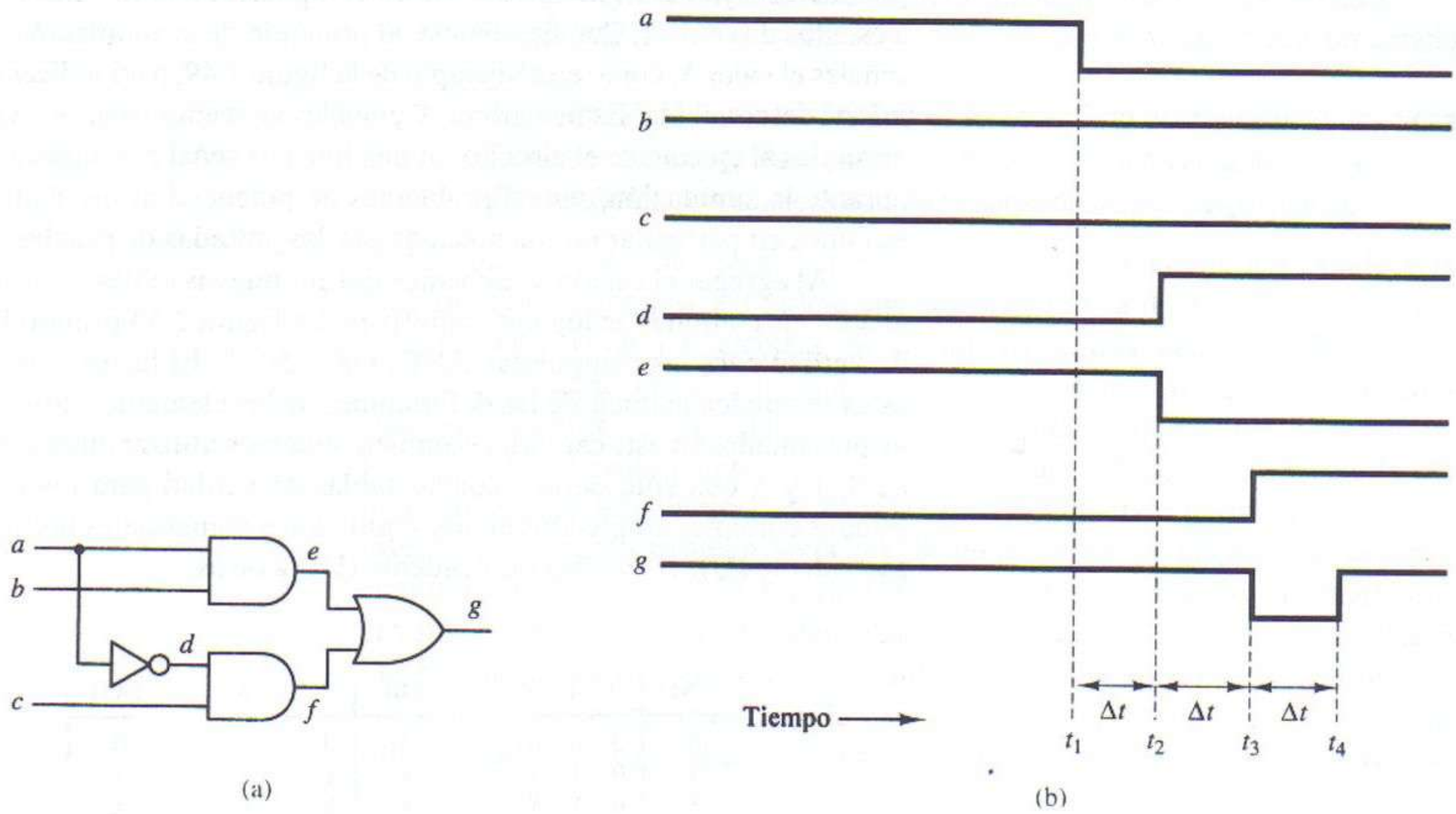


Figura 2.52 Riesgo estático detectado mediante simulación. **(a)** Red con riesgo estático. **(b)** Simulación de formas de onda de salida.

por tanto, tampoco cambiarán. Observe que podemos calcular el tiempo de la simulación global como

$$T_{\text{simulación}} = \frac{\text{tiempo}}{\text{compuerta}} \times \text{compuertas evaluadas}$$

En la simulación controlada por eventos, sólo se vuelven a calcular las salidas de aquellas compuertas afectadas por eventos de entrada, lo que mejora el desempeño global en la mayor parte de los casos, compensando con creces el tiempo de evaluación más lento de cada compuerta. Además, muchas simulaciones compiladas imponen límites a los tipos de modelos de retardo que es posible usar, lo que no ocurre en la simulación controlada por eventos.

Valores de señales lógicas simbólicos

En el capítulo 1 definimos 0 y 1 como los únicos valores posibles de una señal lógica digital. Al evaluar un circuito electrónico real, el diseñador podría necesitar otra información acerca de una señal dada además de su valor lógico, como su intensidad o si va en ascenso o descenso. Por esta razón, los simuladores lógicos proporcionan con frecuencia "valores" de señales distintos de 0 y 1 para poder representar varias condiciones. En estos simuladores, el valor de una señal lógica se representa mediante un *estado* y una *intensidad*, donde el *estado* representa una condición particular y la *intensidad* proporciona información relativa a la compuerta que suministra la señal.

El conjunto mínimo de estados utilizados en la simulación lógica es {0, 1}. Muchos simuladores agregan un tercer valor de estado, X, para representar un estado desconocido o un problema de potencial en la línea de señal, como un

pico de voltaje o el hecho de que varias compuertas intenten llevar una única red a estados diferentes. Con frecuencia, al principio de la simulación se asigna a las señales el valor X, como en el ejemplo de la figura 2.49, para indicar una condición inicial desconocida. Estos valores X iniciales se reemplazan por valores lógicos normales al ejecutarse el circuito. Si una línea de señal permanece con el valor X durante la simulación, indica problemas de potencial al diseñador, puesto que esa línea en particular no fue afectada por las entradas de prueba.

Al agregar el estado X, debemos definir nuevas tablas de verdad para cada una de las compuertas lógicas primitivas. La figura 2.53 proporciona las tablas de verdad para las compuertas AND, OR y NOT. El lector debe verificar que éstas se pueden deducir de las definiciones de los elementos nulo y de identidad ya presentadas en este capítulo. También podemos utilizar otros estados distintos de 0, 1 y X con sólo definir nuevas tablas de verdad para los elementos. Los estados comunes empleados en los simuladores comerciales incluyen el cambio ascendente (U), el cambio descendente (D), y otros.

AND	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

OR	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

NOT	0
0	1
1	0
X	X

Figura 2.53 Tablas de verdad para la lógica con tres valores.

En algunos circuitos, podemos conectar las salidas de varias compuertas a una sola línea, como se muestra en la figura 2.54. En estos casos, debe tomarse

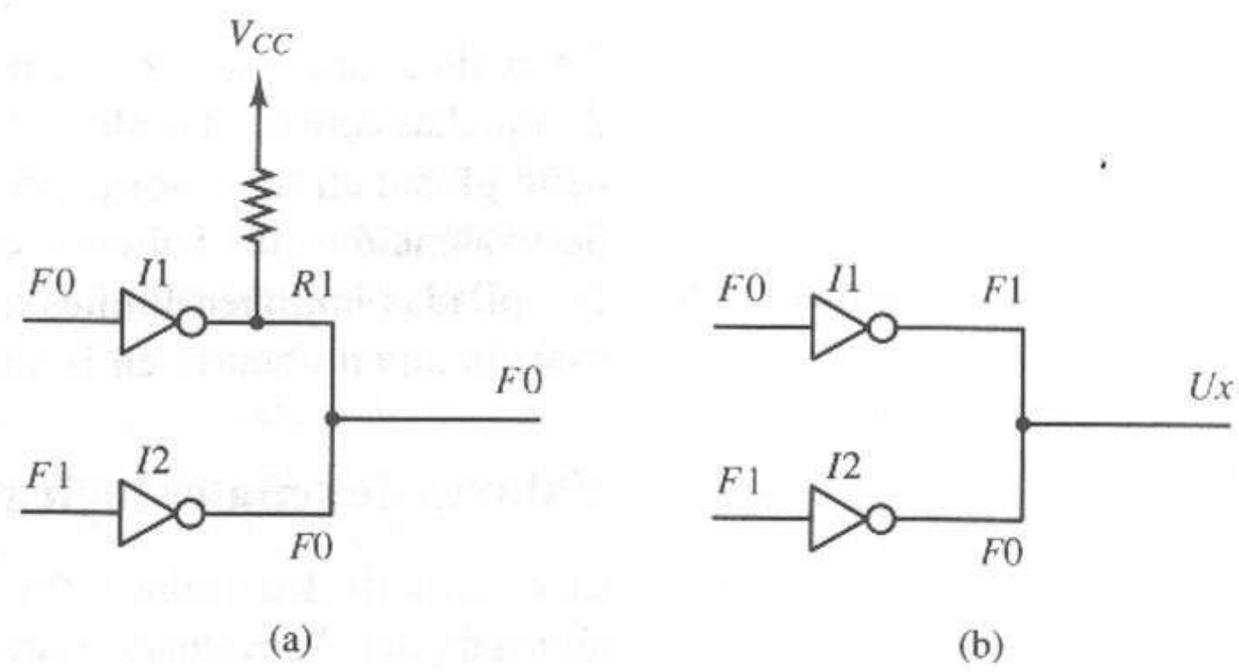


Figura 2.54 Intensidad de señal utilizada para resolver el conflicto entre las salidas de compuerta. **(a)** Salida resuelta a favor de la señal más fuerte. **(b)** Valor de salida sin solución.

en cuenta la *intensidad* con que cada compuerta controla la línea de la señal, de modo que se pueda determinar el valor de señal resultante en la línea, sobre todo si las compuertas intentan llevar la misma línea a estados diferentes. Entre los valores de intensidad de señal comunes utilizados en la simulación están:

Forzada (*F*): la línea de señal se lleva con fuerza hacia un estado dado.
Resistiva (*R*): la línea de señal se lleva con suavidad hacia un estado dado.
Flotante (*Z*): la línea de señal no es forzada. Esto ocurre cuando una línea de señal no es controlada por la salida de ninguna compuerta lógica.
Desconocida (*U*): la intensidad de la señal no puede determinarse.

Cada valor de señal se especifica mediante una combinación del estado y la intensidad. Por ejemplo, $F0$ indica una señal forzada a 0, $R1$ indica una señal llevada suavemente a 1, etcétera. Si las salidas de dos elementos de circuito diferentes se conectan a la misma línea de señal e intentan forzar la línea a estados diferentes, se utilizan los valores de intensidad para resolver el conflicto. Por ejemplo, consideremos el circuito de la figura 2.54a, donde la salida del inversor de colector abierto $I1$ es $R1$ y la salida del inversor normal $I2$ es $F0$. En este caso, el estado resuelto sería $F0$. Si un valor no se puede resolver, por ejemplo, si una salida es $F0$ y la otra es $F1$, como en la figura 2.54b, asignaríamos el valor Ux, lo que indica una intensidad y estado desconocidos. Esta condición avisa al diseñador de que existe un problema de potencial con este circuito.

Si no es importante analizar la intensidad de la señal, podemos especificar sólo el estado, y se supondrá por lo general que F es la intensidad predeterminada.

Modelos de retardo de dispositivos primitivos

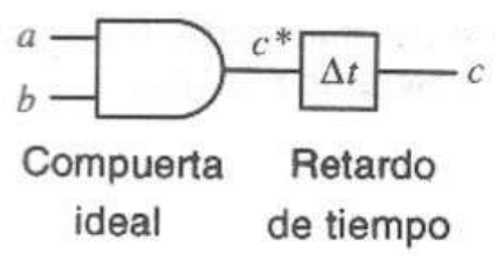


Figura 2.55 Compuerta AND modelada como una compuerta ideal y un retardo por transporte.

Cada compuerta lógica primitiva se caracteriza por una función lógica y un retardo intrínseco, es decir, el tiempo que tarda su salida en responder a un evento de entrada. Los retardos son funciones de la complejidad del circuito, de la tecnología electrónica utilizada, y de factores como el *fan-out* de la compuerta (el número de entradas de otras compuertas controladas por una salida de esta compuerta), la temperatura, el voltaje del circuito, etcétera. La figura 2.55 muestra un modelo típico para una compuerta AND primitiva que consta de una compuerta AND ideal (cero retardo) seguida de un elemento de retardo. Para cada cambio en la salida, el cambio real se retarda en un tiempo Δt, que es el valor asignado al elemento de retardo.

Al procesar un evento en el instante T en una entrada de una compuerta AND, el simulador calcularía primero la salida de la compuerta ideal, c^*, con una tabla de verdad o algún otro modelo. Entonces, se programaría el valor calculado de c^* para aparecer en la salida c del elemento de retardo en el instante $T + \Delta t$. Este comportamiento se conoce a veces como *retardo por transporte*, puesto que la forma de onda de salida en c se vería transportada, o desplazada en el tiempo, por Δt desde la salida de la compuerta ideal c^*.

El elemento de retardo puede incorporar otros modelos de retardo para representar diferentes características físicas de la compuerta. Los modelos de retardo más comunes son los de retardos unitario, nominal, ascenso/descenso, y mín/máx, los cuales describiremos en las siguientes secciones.

Retardo unitario/nominal

La verificación de que un diseño es correcto desde el punto de vista lógico no requiere información detallada de los tiempos. En tal caso, basta suponer que cada compuerta tiene cierto retardo fijo asociado a ella. El método más sencillo es asignar a cada compuerta del circuito el mismo *retardo unitario*, es decir, suponer que cada salida de compuerta responde a un evento de entrada en exacta-

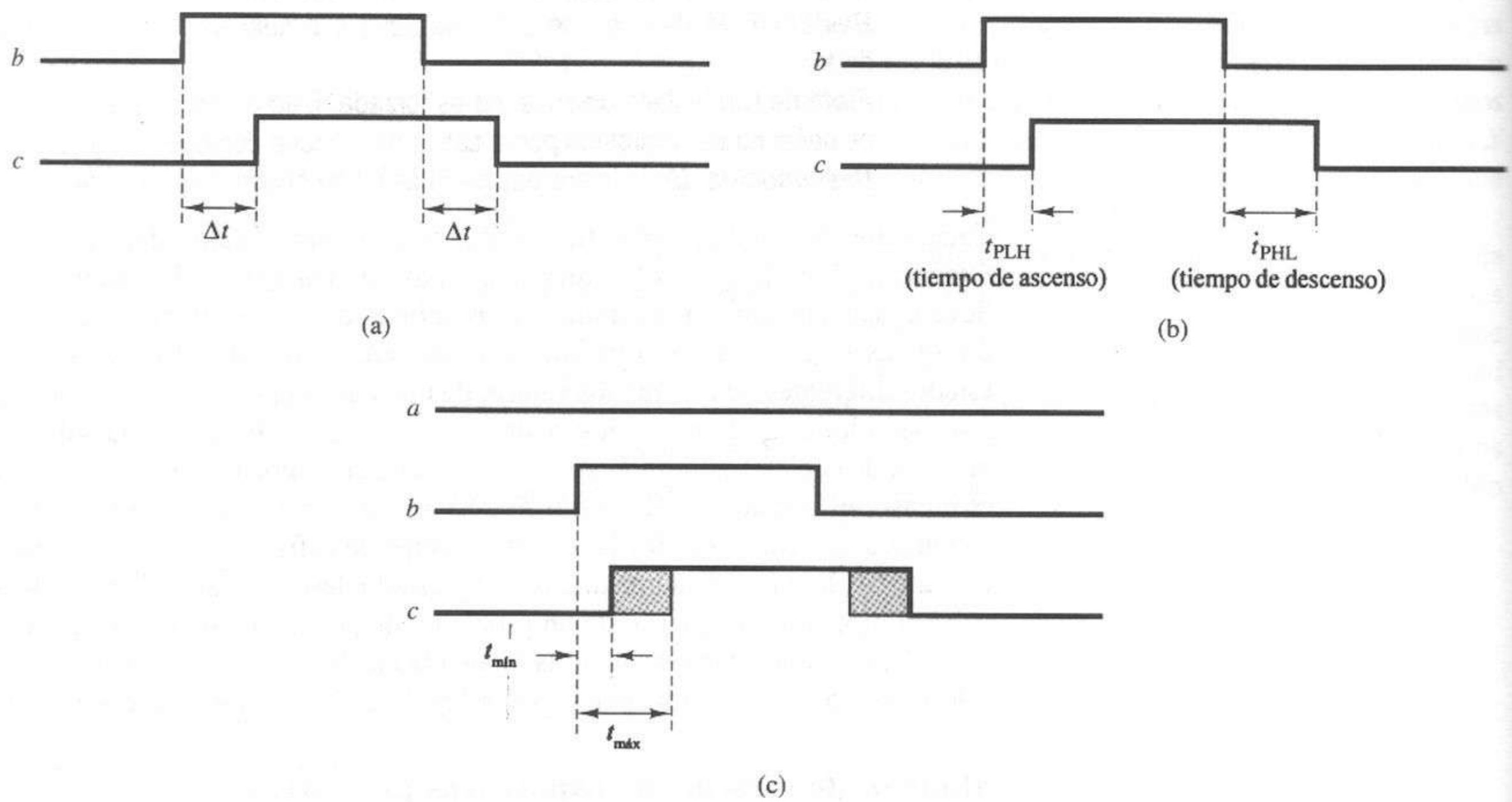


Figura 2.56 Efectos de diversos modelos de retardos por transporte. **(a)** Retardo unitario/nominal. **(b)** Retardo de ascenso/descenso. **(c)** Retardo mín/máx.

exactamente una unidad de tiempo. Entonces, todos los tiempos de simulación se miden en términos de un número entero de unidades de tiempo. El lapso real representado por una unidad de tiempo no es importante para el simulador; el diseñador podría simplemente multiplicar un número dado de unidades de tiempo por el tiempo físico representado por una unidad.

Puesto que las diversas compuertas lógicas tienen características diferentes, se logra una mejor precisión asignando *retardos nominales*, que son retardos por transporte determinados individualmente para cada tipo de compuerta. Por ejemplo, una compuerta simple, como una NAND o NOR, podría tener un retardo nominal de una unidad de tiempo, mientras que un elemento más complejo, como una compuerta XOR, podría tener un retardo nominal de dos unidades de tiempo. La figura 2.56a muestra el funcionamiento de la compuerta AND de la figura 2.55, suponiendo un retardo nominal de Δt para esta compuerta.

Un caso degenerado de los modelos de retardo unitario y nominal es el modelo de *retardo nulo*, en donde se supone que un cambio en la entrada tiene un efecto inmediato sobre la salida. Sin embargo, el modelo de retardo nulo no siempre proporciona una imagen real de la forma de operar de un circuito, sobre todo en el caso de los circuitos secuenciales, como veremos en una sección posterior del texto. Así, el uso de los modelos con retardo nulo se restringe principalmente a la verificación de las ecuaciones lógicas realizadas por un circuito combinatorio.

Retardo de ascenso/descenso

Las salidas de muchos circuitos de compuertas lógicas tardan tiempos distintos en ascender de 0 a 1 que en descender de 1 a 0. Modelamos esto reemplazando el retardo unitario o nominal por un *tiempo de ascenso*, t_{PLH} (retardo por propagación de bajo a alto) y un *tiempo de descenso*, t_{PHL} (retardo por propagación de alto a bajo) por separado. Al programar un evento de salida, el simulador retarda la señal en t_{PLH} para un cambio $0 \rightarrow 1$ y en t_{PHL} para un cambio $1 \rightarrow 0$. Así, un evento $0 \rightarrow 1$ que ocurre en el instante T se programaría en $T + t_{PLH}$, mientras que un evento $1 \rightarrow 0$ que ocurre en el instante T se programaría en $T + t_{PHL}$. La figura 2.56b muestra esto para el modelo de compuerta AND.

Retardos ambiguos o mín/máx

Las características de los dispositivos electrónicos reales son afectadas por las variaciones en el proceso de fabricación o por factores como el voltaje del circuito, la temperatura, y el *fan-out*. Esto hace imposible predecir los tiempos de ascenso o descenso exactos de una señal. En las aplicaciones en que el tiempo es crítico, los diseñadores realizan con frecuencia análisis del peor caso de un circuito para determinar los efectos del hecho de que las compuertas trabajen a máxima o mínima velocidad. Esto se hace especificando un intervalo de valores $\{t_{mín}, t_{máx}\}$ para cada parámetro de tiempos, donde $t_{mín}$ es el retardo mínimo y $t_{máx}$ es el retardo máximo. Cada área sombreada de la figura 2.56c indica el intervalo de tiempo durante el cual podría ocurrir el evento de salida, comenzando el intervalo en el instante $T + t_{mín}$ y terminando en $T + t_{máx}$. La salida puede cambiar en cualquier momento dentro de este intervalo.

Un problema con el uso de los retardos mín/máx es que los resultados tienden a ser pesimistas. Esto se debe a que los resultados indican el peor de los casos de todas las compuertas cuando trabajan con sus retardos más lentos o más rápidos. Es raro que esto ocurra en los circuitos reales.

Por ejemplo, consideremos el circuito de la figura 2.57a y supongamos que cada compuerta tiene un retardo mínimo $t_{mín} = 2$ y un retardó máximo $t_{máx} = 5$. Simulamos el circuito con las condiciones iniciales de las entradas iguales a $a = b = d = 0$ y $c = 1$, para después cambiar la entrada d de 0 a 1 en el instante $t = 10$. La figura 2.57b muestra la forma de onda de salida producida por el simulador empleando los modelos de retardo mín/máx de las compuertas. Observe que el ancho de la *región de ambigüedad*, es decir, la región donde puede ocurrir un cambio, es mayor en cada nivel sucesivo del circuito. El tiempo mínimo en que puede ocurrir un cambio de la salida h es igual al tiempo del cambio de entrada más la suma de los retardos mínimos para las señales e, g y h, es decir, $10 + 3 \times t_{mín} = 16$. De manera similar, el tiempo máximo que tardaría la salida h en cambiar se determina mediante los retardos máximos de las compuertas a lo largo de la trayectoria desde la entrada hasta la salida, es decir, $10 + 3 \times t_{máx} = 25$. Así, la salida podría cambiar en cualquier momento dentro del intervalo [16, 25]. En la realidad, es poco probable que todos los retardos de compuertas tengan el extremo máximo o mínimo. Por tanto, generalmente se espera que el cambio real ocurra en algún momento cercano al punto medio de esta región. Sin embargo, el diseñador debe considerar ambos extremos para garantizar que el circuito opere de manera adecuada en todas las condiciones.

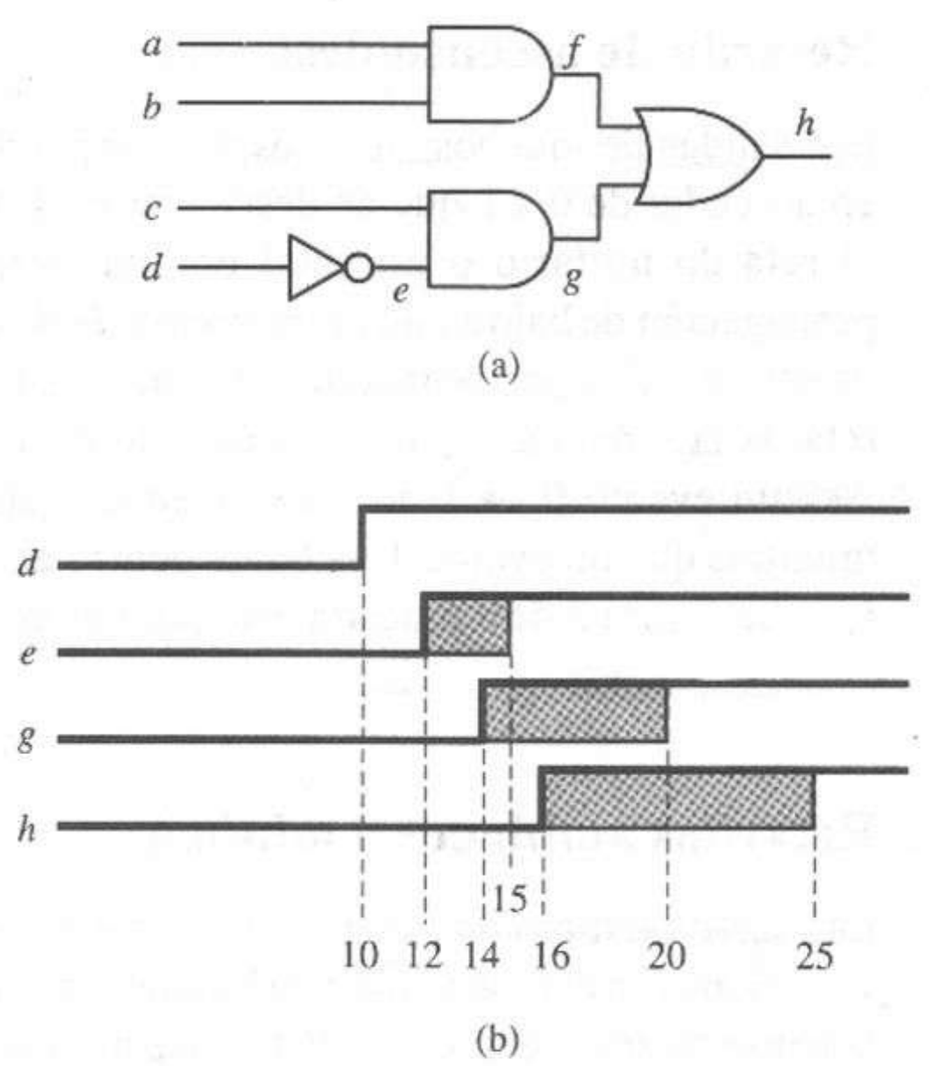


Figura 2.57 Simulación que ilustra la acumulación de retardos min/máx. **(a)** Modelo del circuito. **(b)** Retardos en el peor caso.

Otro método para analizar los retardos mínimo y máximo es efectuar un *análisis de ruta crítica*. En este tipo de análisis, no se realiza una simulación, sino que se identifica cada posible ruta de la señal entre las entradas y las salidas del circuito. Entonces, se suman los retardos mínimo y máximo de todas las compuertas a lo largo de todas las rutas para determinar las respuestas de salida máxima y mínima. Las rutas cuyos retardos máximo y mínimo representan problemas de potenciales se identifican como *rutas críticas* a través del circuito. El diseñador puede entonces modificar el diseño a lo largo de las rutas críticas a fin de evitar estos problemas.

Retardo inercial

Los modelos de retardo descritos son ejemplos de *retardos por transporte*; es decir, se "transporta" un nuevo valor a la salida de la compuerta después del retardo designado. Esto no siempre modela con precisión el funcionamiento de un dispositivo electrónico físico. En el caso de muchos dispositivos, un valor de entrada debe persistir durante cierto tiempo mínimo para proporcionar a la salida la inercia necesaria para cambiar. En tales casos, los picos breves en las entradas no afectan el resultado. La duración mínima de la señal de entrada necesaria para producir un cambio en la salida se conoce como *retardo inercial*.

El efecto del retardo inercial sobre el funcionamiento de un circuito se muestra en los diagramas de tiempos de la figura 2.58. En la figura 2.58a, se supone un modelo de retardo por transporte. Cualquier cambio de entrada produce un cambio correspondiente en la salida después de Δt segundos, sin importar lo breve que sea la duración del valor de entrada. En la figura 2.58b, se supone un retardo inercial de Δt segundos, lo que implica que cualquier cambio de entrada con una duración menor que Δt segundos no afectará la salida. En este caso, dos de los cambios de entrada no producen cambios correspondientes en la salida.

Con frecuencia, los retardos inerciales se modelan como retardos en las entradas de una compuerta, como muestra la figura 2.59. Para un cambio en la

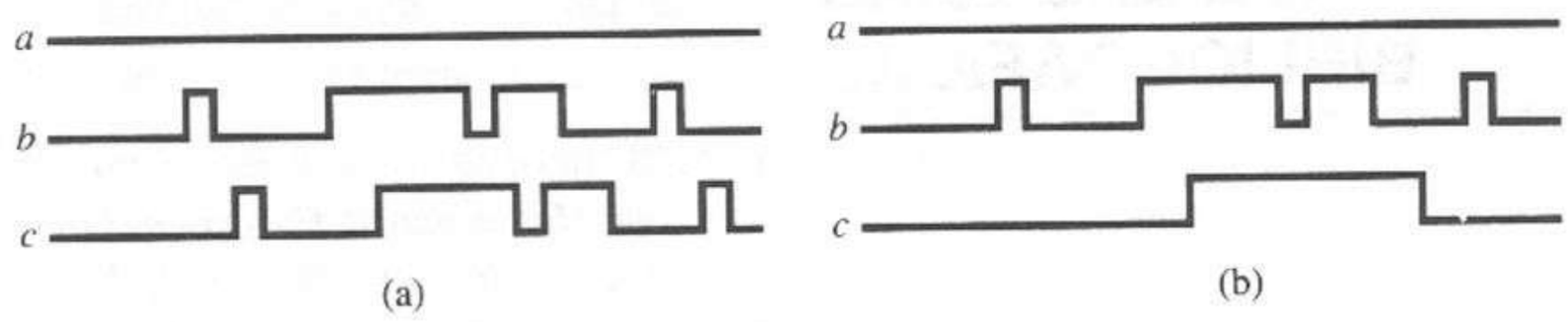


Figura 2.58 Modelos de retardo: transporte *vs.* inercial. **(a)** Modelo de retardo por transporte. **(b)** Modelo de retardo inercial.

entrada a en el instante T, se programa un evento para la línea a^* en el instante $T + \Delta t$. Si ocurre un segundo cambio en la línea a antes del instante $T + \Delta t$, el evento de la línea a^* se cancelará eliminándolo de la cola de eventos. El resultado neto es que no se programa cambio alguno para la salida c de la compuerta.

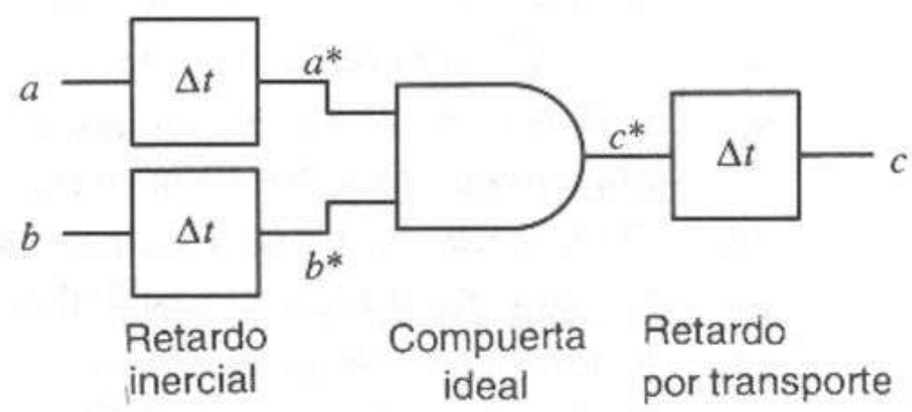


Figura 2.59 Modelo de retardo inercial implantado mediante retardos en las entradas.

2.8 Resumen

El álgebra booleana es la base del análisis y la síntesis de los circuitos de conmutación; por tanto, dedicamos mucho tiempo al desarrollo del álgebra booleana como una herramienta familiar. Además, analizamos los conceptos de diagramas de Venn, tablas de verdad, mintérminos, maxtérminos, etcétera, a fin de que el lector tenga las bases necesarias para enfrentar los problemas del análisis y la síntesis de las redes lógicas combinatorias.

Además, en este capítulo examinamos los elementos básicos del diseño asistido por computadora, aplicado a los circuitos lógicos combinatorios. Mostramos varios métodos para modelar los circuitos lógicos, capturándolos en la base de datos de un sistema de CAD, y simulando su operación para verificar el diseño y predecir tiempos y otras características de los circuitos antes de construirlos en realidad. Además, analizamos algunas de las operaciones de síntesis automatizadas que pueden realizarse mediante los sistemas de CAD, incluida la generación de un diseño de circuito a partir de un modelo abstracto.

El tratamiento en este capítulo no se basó en algún sistema o programa de CAD en particular, sino que intentamos proporcionar un panorama sobre los procesos básicos implicados en el diseño asistido por computadora y el análisis de los circuitos lógicos digitales. El lector debe consultar la documentación y material tutorial incluido con las herramientas CAD disponibles para obtener información específica sobre la forma de llevar a cabo las operaciones aquí descritas.

REFERENCIAS BIBLIOGRÁFICAS

1. G. BOOLE, *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probability*, 1849. Reprinted by Dover Publications, Inc., New York, 1954.
2. Y. CHU, *Digital Computer Design Fundamentals*. New York: McGraw-Hill Book Co., 1962.
3. F. J. HILL, AND G. R. PETERSON, *Computer Aided Logical Design with Emphasis on VLSI, 4th ed.* New York: Wiley, 1993.
4. G. A. MALEY, AND J. EARLE, *The Logic Design of Transistor Digital Computers*. Englewood Cliffs, NJ: Prentice Hall, 1963.
5. E. J. MCCLUSKEY, JR., *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965.
6. D. L. DIETMEYER, *Logic Design of Digital Systems, 3rd ed.* Boston: Allyn and Bacon, 1971.
7. *STANDARD GRAPHIC SYMBOLS FOR LOGIC FUNCTIONS: IEEE/ANSI STANDARD 91-1984*, Institute of Electrical and Electronics Engineers, Inc., IEEE Standards Office, 345 East 47th St., New York, NY 10017, 1984.
8. *THE TTL DATA BOOK, VOLUME 2*, Texas Instruments, Inc., Dallas, TX, 1985.
9. J. F. WAKERLY, *Digital Design Principles and Practices, 2nd ed.*, Chapter 2. Englewood Cliffs, NJ: Prentice Hall, 1994.
10. *IEEE STANDARD VHDL LANGUAGE REFERENCE MANUAL*, IEEE Std. 1076-1987, March 1988, IEEE, 345 East 47th Street, New York, NY 10017.
11. R. LIPSETT, C. SCHAEFER, AND C. USSERY, *VHDL: Hardware Description and Design*. Kluwer Academic Publishers, 1989.
12. J. R. ARMSTRONG, *Chip-Level Modeling With VHDL*. Englewood Cliffs, NJ: Prentice Hall 1989.
13. DONALD E. THOMAS, AND PHILIP MOORBY, *The Verilog Hardware Description Language*. Norwell, Massachusetts: Kluwer Academic Publishers, 1991.
14. M. D. EDWARDS, *Automatic Logic Synthesis Techniques for Digital Systems*. New York: McGraw-Hill, 1992.
15. M. ABRAMOVICI, M. A. BREUER, AND A. D. FRIEDMAN, *Digital System Testing and Testable Design*. New York: IEEE Press, 1990.

PROBLEMAS

2.1 Utilice el álgebra de conmutación para simplificar las siguientes expresiones:

(a) $f(w, x, y, z) = x + xyz + \bar{x}yz + wx + \bar{w}x + \bar{x}y$

(b) $f(A, B, C, D, E) = (AB + C + D)(\bar{C} + D)(\bar{C} + D + E)$

(c) $f(x, y, z) = y\bar{z}(\bar{z} + \bar{z}x) + (\bar{x} + \bar{z})(\bar{x}y + \bar{x}z)$

2.2 Simplifique las siguientes expresiones de conmutación.

(a) $f(A, B, C, D) = \overline{(A + \bar{C} + D)(\bar{B} + C)(A + \bar{B} + D)(\bar{B} + C)(\bar{B} + C + \bar{D})}$

(b) $f(A, B, C, D) = \overline{AB + \bar{A}\bar{D} + B\bar{D} + \bar{A}B + C\bar{D}A + \bar{A}D + CD + \bar{A}\bar{B}\bar{D}}$

(c) $f(A, B, C, D) = \overline{A\bar{B}C + AB + \overline{ABC} + A\bar{C} + AB\bar{C}}$

(d) $f(A, B, C) = \overline{(B + \bar{A})(AB + C) + AB\bar{A} + \bar{A}\bar{B}C + (A + B)(\bar{A} + C)}$

(e) $f(A, B, C) = \overline{\overline{(\bar{A} + \bar{B})}(A + \bar{A}B)(\bar{A} + \bar{B} + \bar{A}\bar{B}C) + \overline{(A + B)(\bar{A} + C)}}$

2.3 Demuestre la parte (b) del teorema 4 (absorción).

2.4 Demuestre la parte (b) del teorema 5 (absorción).

2.5 Demuestre la parte (b) del teorema 9 (consenso).

2.6 Simplifique las siguientes expresiones de conmutación.

(a) $f(A, X, Z) = \bar{X}(X + Z) + \bar{A} + AZ$

(b) $f(X, Y, Z) = (\bar{X}Y + XZ)(X + \bar{Y})$

(c) $f(x, y, z) = \bar{x}y(z + \bar{y}x) + \bar{y}z$

2.7 Determine la expresión de conmutación más sencilla para las siguientes funciones.

(a) $f(A, B, C) = \sum m(1, 4, 5)$

(b) $f(A, B, C, D) = \prod M(0, 2, 4, 5, 8, 11, 15)$

(c) $f(A, B, C, D) = \sum m(0, 2, 5, 8, 9, 10, 13)$

2.8 Dada la función $f(x, y, z) = x\bar{y} + x\bar{z}$, escriba $f(x, y, z)$ como una suma de mintérminos y como un producto de maxtérminos.

2.9 Utilice diagramas de Venn para determinar cuáles de las siguientes funciones de conmutación son equivalentes.

$$f_1(A, B, C) = A\bar{B}\bar{C} + B + \bar{A}\bar{B}C$$
$$f_2(A, B, C) = \bar{A}\bar{B}\bar{C} + B + A\bar{B}C$$
$$f_3(A, B, C) = \bar{A}\bar{C} + AC + B\bar{C} + \bar{A}B$$
$$f_4(A, B, C) = A\bar{C} + AB + B\bar{C} + \bar{A}C$$

2.10 Trace las siguientes funciones sobre un diagrama de Venn.

(a) $f(A, B) = AB + \bar{A}\bar{B}$

(b) $f(A, B, C) = AB + \bar{A}\bar{C}$

(c) $f(A, B, C, D) = A + \bar{B}CD + \bar{A}BD$

(d) $f(A, B, C, D) = \bar{A}B + C\bar{D}$

Sugerencia: Cada nueva variable se representa mediante un contorno que divide cada segmento del diagrama de Venn en dos segmentos. La figura P2.10 muestra un diagrama de Venn con cuatro variables.

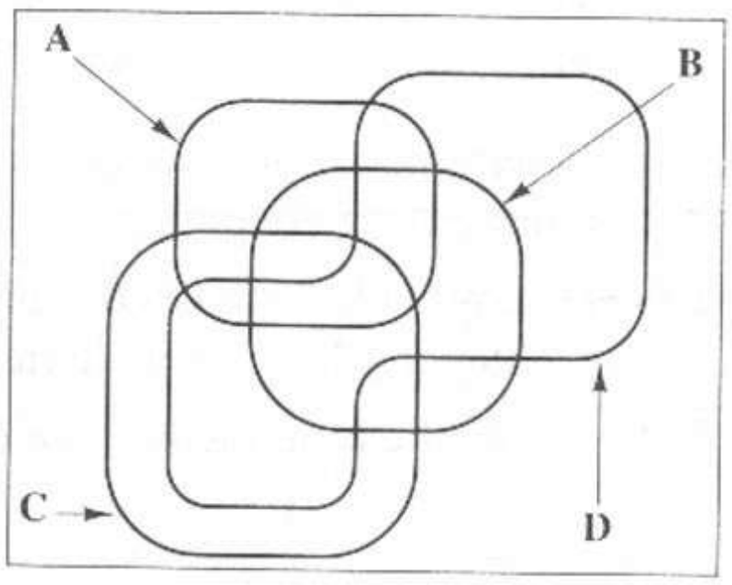


Figura P2.10

2.11 Demuestre que las siguientes expresiones son válidas, utilizando diagramas de Venn.

(a) $A + B = A\bar{B} + \bar{A}B + AB = \overline{\bar{A}\bar{B}}$

(b) $A\bar{C} + BC + A\bar{B} \neq \bar{B}\bar{C} + \bar{A}B + AC$

(c) $\bar{A}C + AB + \bar{B}\bar{C} = \bar{A}\bar{B} + BC + A\bar{C}$

(d) $AD + A\bar{C}\bar{D} + AB + \bar{A}BD + \bar{A}\bar{B}\bar{C} = AB + BD + AD + \bar{B}\bar{C}$

2.12 Utilice el teorema 5 para simplificar las siguientes expresiones.

(a) $\bar{X} + XAB\bar{C} + \bar{B}C$

(b) $\bar{X}\bar{Y} + (X + Y)Z$

(c) $Z(\bar{Z} + AB\bar{C}) + \bar{A}\bar{B}$

(d) $(\bar{X} + \bar{Y})(XY + Z)$

2.13 Emplee el teorema 8 (DeMorgan) para complementar las siguientes expresiones:

(a) $X(Y + \bar{Z}(Q + \bar{R}))$

(b) $X + Y(\bar{Z} + Q\bar{R})$

(c) $XY + A\bar{C} + IQ$

(d) $(A + B\bar{C})(\bar{A} + \bar{D}E)$

2.14 Aplique el teorema 9 del álgebra de conmutación (consenso) para simplificar las siguientes expresiones:

(a) $QR + \bar{X}Q + RX$

(b) $(X + Y)Z + \bar{X}\bar{Y}W + ZW$

(c) $(\bar{X} + Y)WZ + X\bar{Y}V + VWZ$

(d) $(X + Y + Z + \bar{W})(V + X)(\bar{V} + Y + Z + \bar{W})$

2.15 Utilice el teorema 10 (teorema de desarrollo de Shannon) para transformar las siguientes funciones al formato

$$f(A, B, C, Q) = \bar{Q}f_\alpha(A, B, C) + Qf_\beta(A, B, C)$$
$$= [\bar{Q} + f_\gamma(A, B, C)][Q + f_\delta(A, B, C)]$$

Determine f_α, f_β, f_γ y f_δ cuando

(a) $f(A, B, C, Q) = (Q + \bar{A})(\bar{B} + C) + \bar{Q}\bar{C}$

(b) $f(A, B, C, Q) = A\bar{B}\bar{C} + Q\bar{A} + \bar{Q}C$

(c) $f(A, B, C, Q) = (A + \bar{B} + Q)(\bar{A} + \bar{Q} + C)$

(d) $f(A, B, C, Q) = AB\bar{C} + \bar{A}C$

2.16 Determine las tablas de verdad de las siguientes funciones de conmutación.

(a) $f(A, B) = A + \bar{B}$

(b) $f(A, B, C) = AB + \bar{A}\bar{C}$

(c) $f(a, b, c) = a\bar{b}c + b\bar{c}$

(d) $f(a, b, c) = a(b + \bar{c})(\bar{b} + c)$

2.17 Determine las tablas de verdad de las siguientes funciones de conmutación.

(a) $f(A, B, C, D) = AB\bar{C}D + ABC\bar{D}$

(b) $f(A, B, C, D) = AB + \bar{A}\bar{B} + C\bar{D}$

(c) $f(A, B, C, D) = A(\bar{B} + C\bar{D}) + \bar{A}B\bar{C}D$

2.18 Determine las formas de listas de mintérminos y maxtérminos para las funciones de conmutación del problema 2.16. Utilice cualquier método.

2.19 Determine la forma SOP canónica para las funciones de conmutación del problema 2.17. Use cualquier técnica.

2.20 Desarrolle la siguiente función en la forma SOP canónica.

$$f(x_1, x_2, x_3) = x_1\bar{x}_3 + x_2\bar{x}_3 + x_1x_2x_3$$

2.21 Desarrolle la siguiente función en la forma POS canónica.

$$f(W, X, Q) = (Q + \bar{W})(X + \bar{Q})(W + X + Q)(\bar{W} + \bar{X})$$

2.22 Una alarma contra robos está diseñada de modo que percibe cuatro líneas de señal de entrada. La línea A es del interruptor secreto de control, la línea B es del sensor de presión bajo una caja fuerte en un gabinete cerrado,

la línea C es de un reloj alimentado por baterías y la línea D está conectada a un interruptor en la puerta cerrada del gabinete. Las siguientes condiciones producen un voltaje de 1 lógico en cada línea.

A: El interruptor de control está cerrado.

B: La caja está en su posición normal en el gabinete.

C: El reloj marca entre las 1000 y las 1400 horas.

D: La puerta del gabinete está cerrada.

Escriba la expresión de conmutación para la alarma contra robos que produce un 1 lógico (suena un timbre) cuando la caja se mueve y el interruptor de control está cerrado, o cuando el gabinete se abre después de horas hábiles, o cuando el gabinete está abierto con el interruptor de control abierto.

2.23 Un corredor largo tiene tres puertas, una en cada extremo y una a la mitad. Cada puerta tiene un interruptor para operar las luces a lo largo del pasillo. Rotule los interruptores como A, B y C. Diseñe una red lógica para controlar las luces.

2.24 Determine el circuito mínimo equivalente al que se muestra en la figura P2.24.

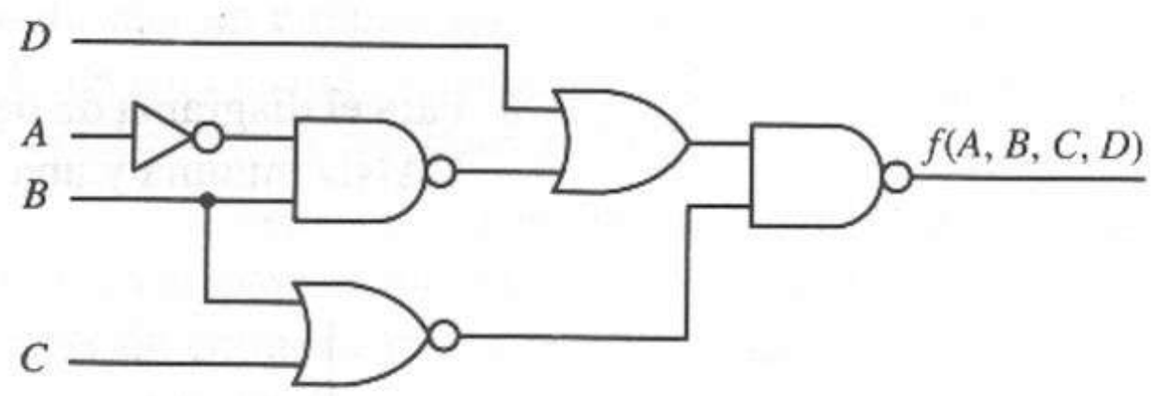


Figura P2.24

2.25 Dado el diagrama de tiempos de la figura P2.25, determine la expresión de conmutación más sencilla para $f(A, B, C)$.

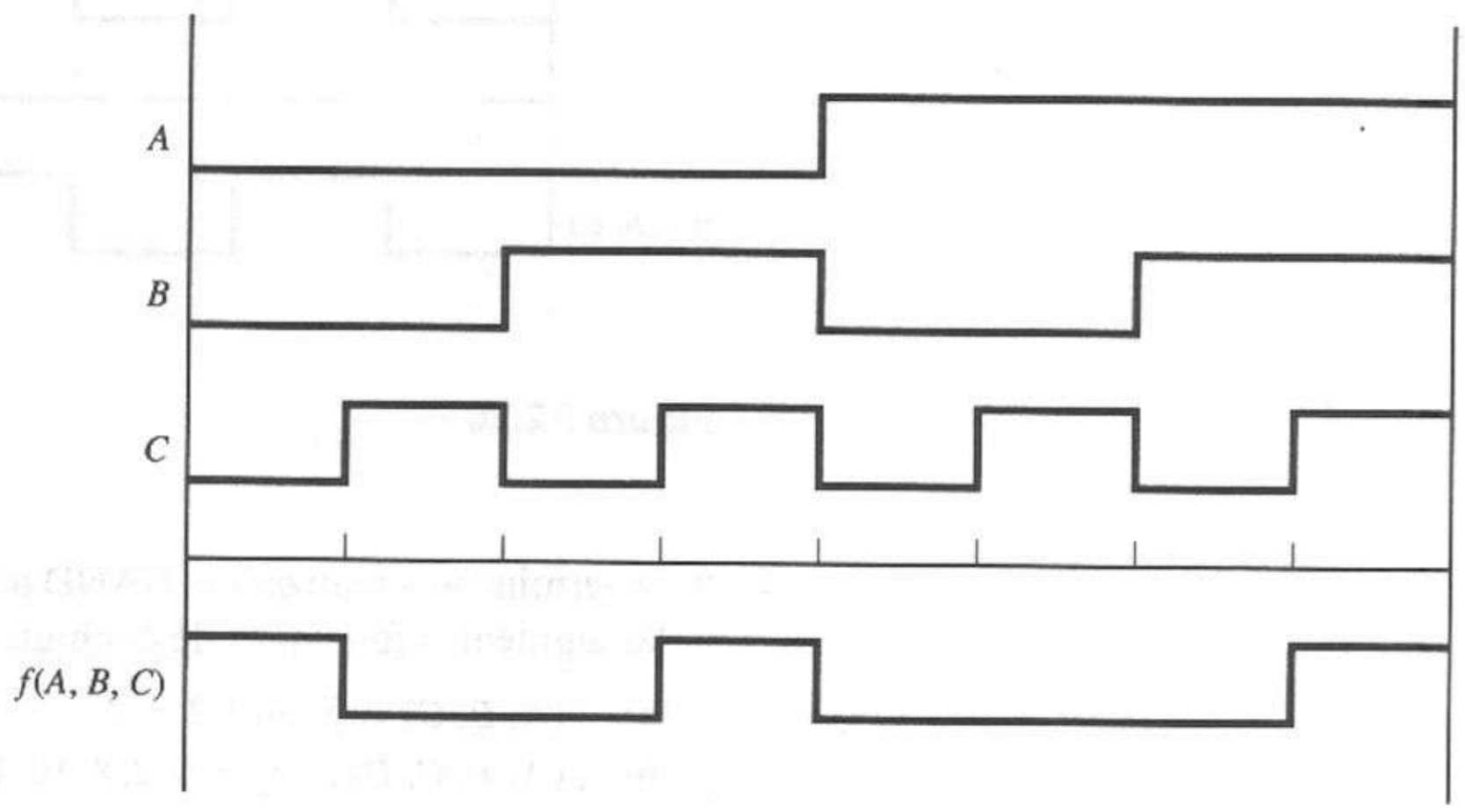


Figura P2.25

2.26 Determine una realización NOR mínima de dos niveles para la siguiente función de conmutación.

$$f(A, B, C) = \sum m(1, 2, 3, 5, 6, 7, 8, 9, 12, 14)$$

2.27 Dada la red que se muestra en la figura P2.27, determine la realización NOR mínima de dos niveles.

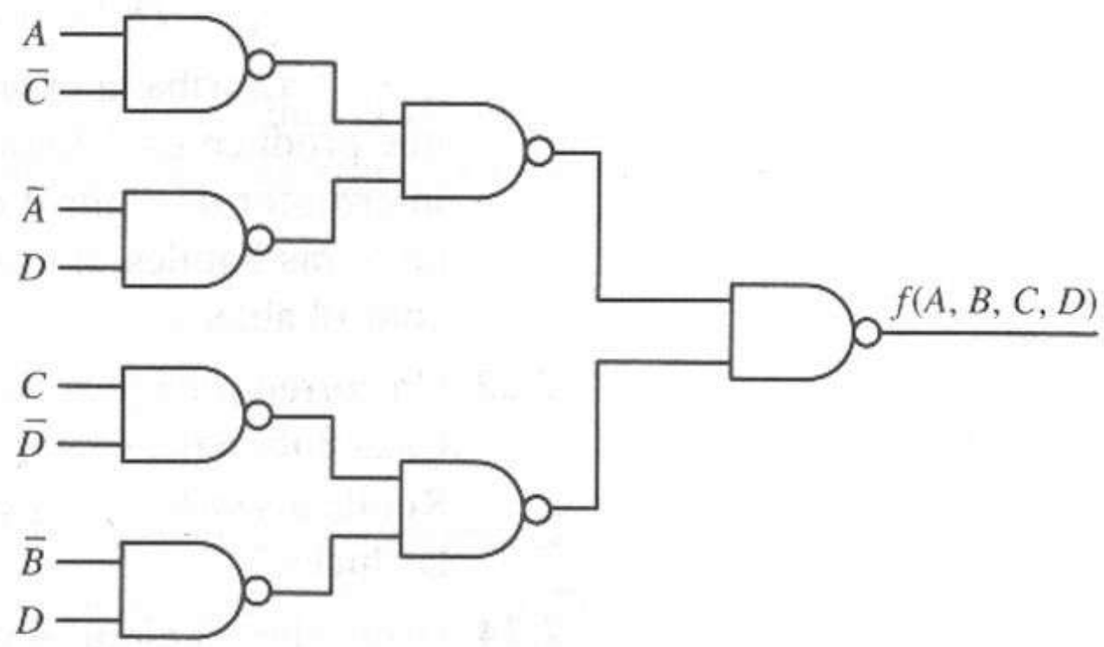


Figura P2.27

2.28 Para el diagrama de tiempos de la figura P2.28, determine una realización NAND mínima y una NOR mínima.

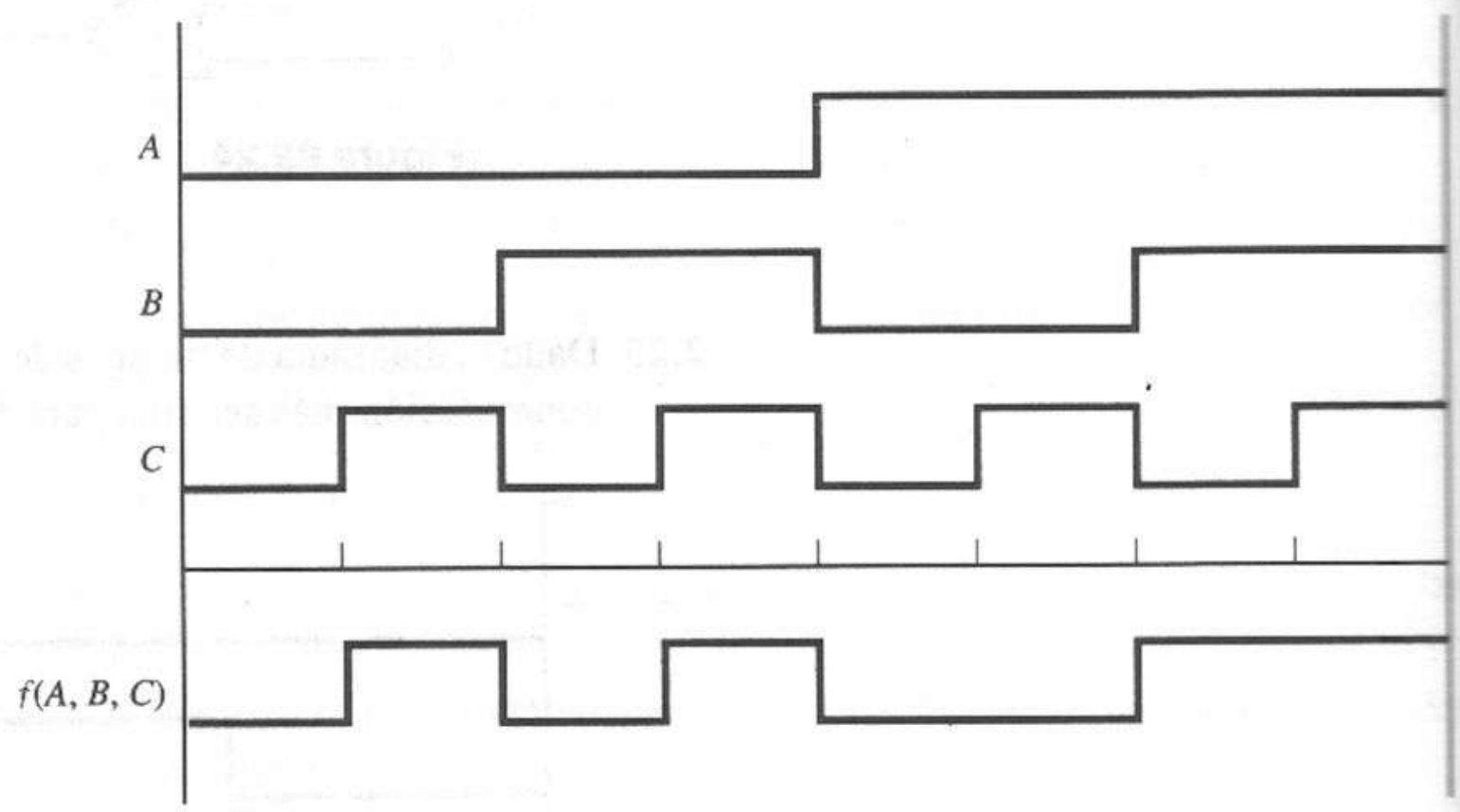


Figura P2.28

2.29 Determine una realización NAND mínima de dos niveles para cada una de las siguientes funciones de conmutación.

(a) $f(A, B, C) = \sum m(0, 2, 3, 7)$

(b) $f(A, B, C, D) = \sum m(0, 2, 8, 10, 14, 15)$

(c) $f(A, B, C, D, E) = \sum m(4, 5, 6, 7, 25, 27, 29, 31)$

2.30 Obtenga una realización NOR mínima de dos niveles para las siguientes funciones de conmutación.

(a) $f(A, B, C) = \sum m(0, 2, 3, 7)$

(b) $f(A, B, C, D) = \sum m(0, 2, 8, 10, 14, 15)$

(c) $f(A, B, C, D, E) = \sum m(0, 1, 2, 3, 8 \text{ to } 24, 26, 28, 30)$

2.31 José, Juan y Pedro se reúnen una vez a la semana para ir al cine o al boliche. Para decidir qué hacer, votan y gana la mayoría simple. Si un voto por el cine se representa mediante un 1, diseñe un circuito lógico que calcule en forma automática la decisión.

2.32 Deduzca las ecuaciones lógicas para un circuito que reste dos números binarios de 2 bits, $(X_1X_0)_2 - (Y_1Y_0)_2$ y produzca como salida el número resultante $(D_1D_0)_2$ y la condición de préstamo B_1.

$$\begin{array}{r} X_1X_0 \\ -Y_1Y_0 \\ \hline B_1D_1D_0 \end{array}$$

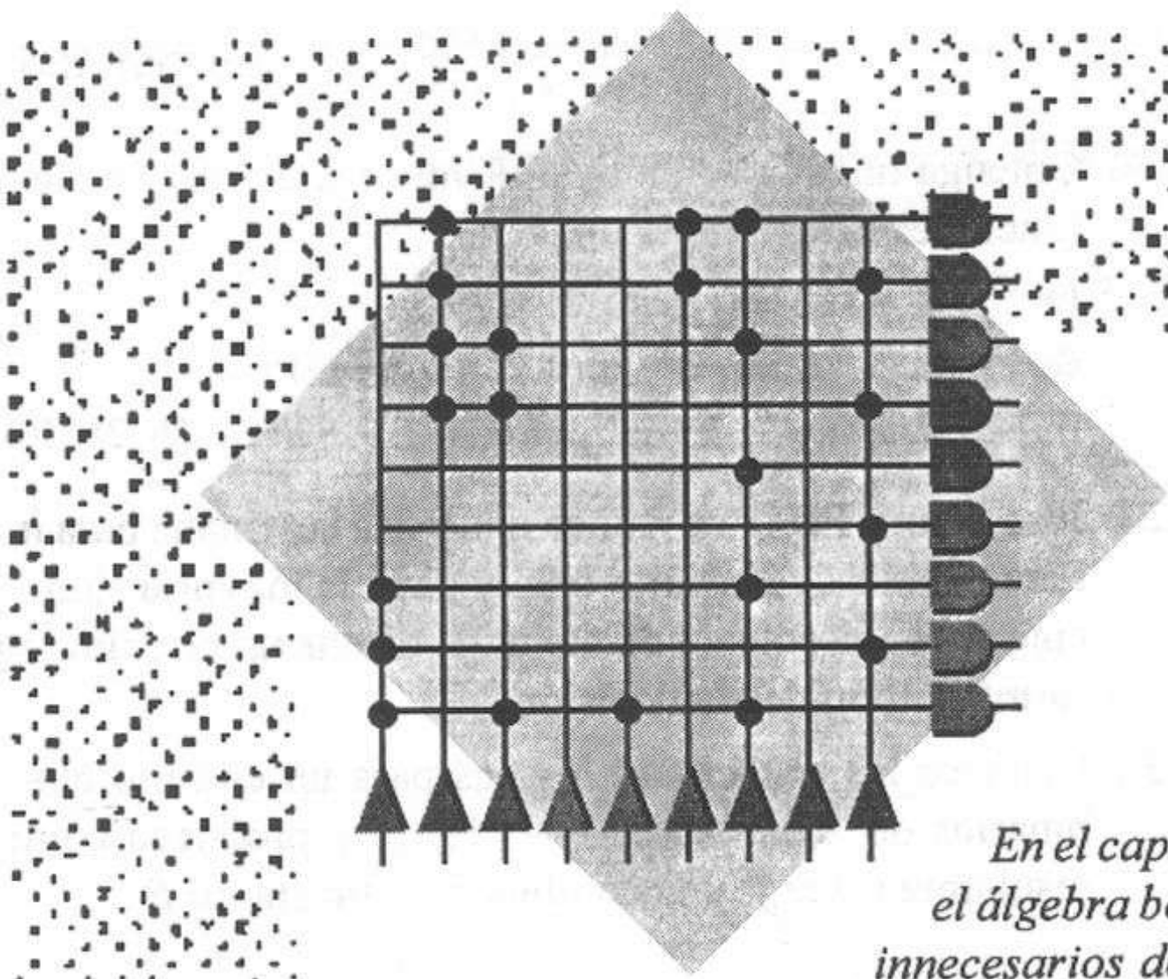
2.33 Deduzca la ecuación lógica y el diagrama de circuito para un circuito con tres entradas *A, B* y *C*. La salida será alta sólo cuando exactamente una de las entradas sea alta. Utilice sólo compuertas NAND en el diseño.

2.34 Queremos diseñar un circuito lógico con cuatro entradas *A, B, C* y *D*. La salida será alta sólo cuando la mayor parte de las entradas sea alta. Trace el circuito final sólo con compuertas NOR.

2.35 Un circuito lógico tiene cuatro entradas, *A, B, C* y *D*. Determine las ecuaciones lógicas para el circuito, si la salida será alta sólo cuando un número impar de entradas sean altas. Trace un diagrama de circuito con las compuertas que desee.

2.36 La entrada de un circuito lógico consta de cuatro líneas de señal *A, B, C* y *D*. Estas líneas representan un número binario de 4 bits, donde *A* representa el bit más significativo y *D* el menos significativo. Diseñe el circuito lógico de modo que la salida sea alta sólo cuando la entrada binaria sea menor que $(0111)_2 = 7_{10}$. Utilice las compuertas que desee.

3



En el capítulo 2 aprendimos cómo se puede utilizar el álgebra booleana para eliminar términos y literales innecesarios de una expresión de conmutación, lo que permite trabajar con un número mínimo de compuertas lógicas. Por desgracia, este manejo algebraico depende mucho de la habilidad del usuario para identificar los puntos donde conviene aplicar los diferentes postulados y teoremas, haciendo que este proceso sea propenso a errores y poco práctico para la mayor parte de las funciones de conmutación, excepto las más sencillas. En este capítulo examinaremos varios métodos para automatizar la minimización de las funciones de conmutación con especificaciones completas o incompletas. Utilizaremos los mapas de Karnaugh para deducir en forma gráfica expresiones mínimas de suma de productos y producto de sumas para las funciones de conmutación, y emplearemos el método tabular de Quine-McCluskey para simplificar funciones de una o varias salidas. Después examinaremos algunos algoritmos de minimización que se pueden programar e incorporar con facilidad en los sistemas de diseño asistido por computadora.

Simplificación de las funciones de conmutación

3.1 Objetivos de la simplificación

La simplificación de las funciones de conmutación es una meta común e importante. Su importancia radica en el hecho de que cuanto más sencilla sea la función, más fácil será realizarla. El objetivo de la simplificación es minimizar el costo de realización de una función mediante elementos de circuito físicos, donde el costo depende de la naturaleza de los elementos de circuito por utilizar. En general, es deseable minimizar el número de elementos de circuito y hacer que cada elemento sea lo más sencillo posible. En la realización de una suma de productos de dos niveles de una función de conmutación, minimizar el costo implica reducir el número de términos producto en la expresión que representa a la función (para reducir el número de compuertas) y minimizar el número de literales en cada término producto (para minimizar la complejidad de la compuerta, medida en términos del número de entradas de la misma). Al utilizar algunos dispositivos lógicos programables (que describiremos en el capítulo 5), el número de entradas de la lógica que forma los términos producto es constante, de modo que la reducción del número de entradas a una compuerta no representa un ahorro de hardware y, por tanto, sólo es significativo el número de términos. En otros casos, podrían preferirse formas algebraicas distintas de la suma de productos o producto de sumas de dos niveles si con ello se puede reducir el número de compuertas. En el diseño de tarjetas de circuitos impresos (como los módulos utilizados dentro de los sistemas de cómputo), el número total de dispositivos de circuitos integrados (IC) podría ser un factor más limitante que el número de las compuertas individuales.

En todos los casos, hay que lograr un diseño ajustado a las restricciones de los elementos de circuito por utilizar. Es posible que los elementos tengan un número limitado de entradas (*fan-in*) o que esté limitado el número de compuertas que su salida puede controlar (*fan-out*). En algunos casos, el diseñador podría estar restringido a un tipo específico de elementos de circuito. Por último, las consideraciones de tiempos podrían imponer el uso de una realización de dos niveles más rápida, en vez de otra más lenta de tres o más niveles. Además, hay que tomar medidas para prevenir los cambios de salida momentáneos no deseables, llamados *riesgos*, que se presentan debido a los retardos por propagación no uniformes en un circuito.

En este capítulo supondremos primero que queremos minimizar el número de compuertas necesarias para una realización de dos niveles (número mínimo de productos en una forma SOP, o número mínimo de sumas en una forma POS). Si encontramos dos o más expresiones con el mismo número de términos,

elegiremos la expresión con el menor número de literales para obtener compuertas con el menor *fan-in*. Recordemos que una *literal* es cada aparición de una variable o de su complemento.

EJEMPLO 3.1

Determinar el número de términos y literales en las siguientes funciones:

$$g(A,B,C) = A\bar{B} + \bar{A}B + AC$$
$$f(X,Y,Z) = \bar{X}Y(Z + \bar{Y}X) + \bar{Y}Z$$

$g(A, B, C)$ es una forma de dos niveles con tres términos producto y seis literales, mientras que $f(X, Y, Z)$ es una forma de cuatro niveles con siete literales, combinadas mediante tres productos y dos sumas.

3.2 Características de los métodos de minimización

Los postulados y teoremas del álgebra de conmutación son los mecanismos que utilizaremos para minimizar el número de términos y literales de una función de conmutación. Disponemos de varios algoritmos para aplicar estos postulados y teoremas de manera metódica. Los métodos presentados en este capítulo son *heurísticos*, es decir, utilizan la información del problema como guía para la solución, lo que con frecuencia permite tomar decisiones arbitrarias cuando no es clara una elección óptima. Así, los métodos heurísticos no siempre garantizan encontrar la solución mínima y, por lo general, se consideran *subóptimos*, aunque la mayor parte de las veces sí determinen soluciones mínimas.

Existen algunos métodos óptimos en forma de algoritmos formales que siempre garantizan la generación de la solución mínima a un problema. Sin embargo, gran parte de estos algoritmos es más compleja y difícil de aplicar que los métodos heurísticos. Por tanto, muchos diseñadores se conforman con emplear métodos heurísticos y prefieren evitar la complejidad sacrificando la optimalidad en la solución.

Los siguientes ejemplos demuestran la aplicación de diversos postulados y teoremas del álgebra de conmutación para simplificar funciones de conmutación. En las siguientes secciones examinaremos algunos métodos para automatizar el proceso de simplificación.

EJEMPLO 3.2

Utilizar el álgebra de conmutación para determinar las formas SOP y POS de la función $f(X, Y, Z)$ del ejemplo 3.1.

Podemos minimizar esta expresión como sigue.

$$\begin{aligned} f(X,Y,Z) &= \bar{X}Y(Z + \bar{Y}X) + \bar{Y}Z \\ &= \bar{X}YZ + \bar{X}Y\bar{Y}X + \bar{Y}Z && [\text{P5(b)}] \\ &= \bar{X}YZ + \bar{Y}Z && [\text{P6(b), P2(a)}] \\ &= \bar{X}Z + \bar{Y}Z && [\text{T7(a)}] \\ &= (\bar{X} + \bar{Y})Z && [\text{P5(b)}] \end{aligned}$$

Las últimas dos formas representan las formas mínimas SOP y POS, respectivamente. La forma SOP mínima tiene dos términos y un total de cuatro literales. Por tanto, requiere dos compuertas AND de dos entradas y una compuerta OR de dos entradas. Por otro lado, la forma POS requiere una compuerta OR de dos entradas y una compuerta AND de dos entradas.

EJEMPLO 3.3

Utilizar el álgebra de conmutación para encontrar una expresión SOP mínima de la función

$$f(A,B,C,D) = ABC + ABD + \bar{A}B\bar{C} + CD + B\bar{D}$$

que tiene cuatro variables y 13 literales.

$$\begin{aligned}
f(A, B, C, D) &= ABC + ABD + \bar{A}B\bar{C} + CD + B\bar{D} \\
&= ABC + AB + \bar{A}B\bar{C} + CD + B\bar{D} && \text{[T7(a)]} \\
&= ABC + AB + B\bar{C} + CD + B\bar{D} && \text{[T7(a)]} \\
&= AB + B\bar{C} + CD + B\bar{D} && \text{[T4(a)]} \\
&= AB + CD + B(\bar{C} + \bar{D}) && \text{[P5(b)]} \\
&= AB + CD + B\overline{CD} && \text{[T8(b)]} \\
&= AB + CD + B && \text{[T5(a)]} \\
&= B + CD && \text{[T4(a)]}
\end{aligned}$$

Observe que hemos reducido el número de literales de 13 a 3.

En los ejemplos anteriores, la optimalidad de las expresiones finales depende de la habilidad del diseñador para determinar el mejor postulado o teorema que conviene aplicar en cada paso de la simplificación. Esta tarea se hace cada vez más difícil al crecer la complejidad de la expresión por simplificar. Los métodos presentados en este capítulo están diseñados para automatizar estos pasos e incrementar así la posibilidad de encontrar las soluciones óptimas de funciones de complejidad arbitraria.

3.3 Mapas de Karnaugh

Anteriormente vimos que la simplificación de las funciones de conmutación a través del álgebra de conmutación es una tarea difícil, en el mejor de los casos. En otras palabras, el álgebra de conmutación no ofrece un mapa con los caminos a seguir y, por tanto, debemos buscar el mejor método posible, como un alpinista que confía en su intuición y su experiencia. Si queremos realizar eficazmente la minimización de las funciones de conmutación, es claro que debemos contar con técnicas viables, estandarizadas y sistemáticas, que nos proporcionen un mapa del camino hacia la respuesta correcta. El mapa de Karnaugh es una herramienta de este tipo, aplicable a funciones de conmutación de hasta seis variables [1, 2].

3.3.1 Relación con los diagramas de Venn y las tablas de verdad

En realidad, el mapa de Karnaugh no es más que una extensión de los conceptos de tablas de verdad, diagramas de Venn y mintérminos. Con el fin de hacer explícita la extensión, transformaremos un diagrama de Venn en un mapa de Karnaugh. Consideremos el diagrama de Venn de la figura 3.1a. Las dos variables

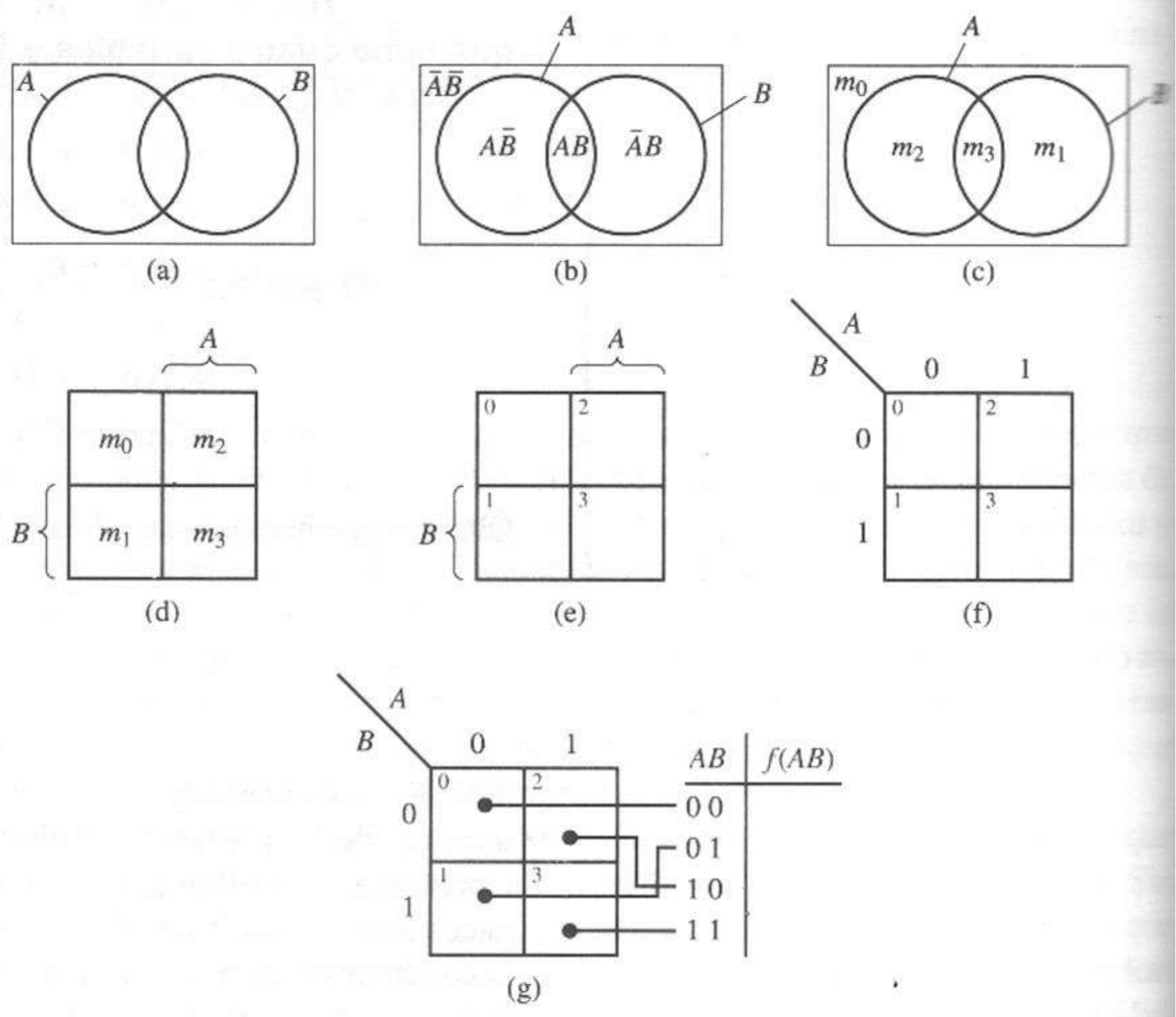


Figura 3.1 Diagrama de Venn y mapa de Karnaugh equivalente para dos variables.

A y B se representan mediante las subdivisiones indicadas del conjunto universal. La figura 3.1b muestra que cada una de las subdivisiones ajenas únicas del diagrama de Venn está formada por una intersección: AB, $\bar{A}B$, $A\bar{B}$ y $\bar{A}\bar{B}$. El lector debe observar que estas intersecciones son sólo los mintérminos de dos variables. Llamaremos a las subdivisiones del diagrama de Venn como los mintérminos: m_0, m_1, m_2 y m_3 (figura 3.1c). Esta forma del diagrama de Venn tiene áreas desiguales para los cuatro mintérminos. Sin embargo, podemos ajustar las áreas y hacer que todas sean iguales, como en la figura 3.1d. Observe que las áreas adyacentes del diagrama de Venn también son adyacentes en la figura 3.1d, pero ahora una mitad del diagrama representa a la variable A y otra mitad representa a B. Puesto que identificamos la notación de mintérminos con cada cuadrado del diagrama, podemos omitir la letra m y dejar sólo el subíndice, como en la figura 3.1e. Ésta es una forma del mapa de Karnaugh. Una segunda forma del mapa de Karnaugh aparece en la figura 3.1f. En esta última forma, la

asociación de un cuadrado de un mapa con una variable en particular, por ejemplo A, se indica como 0 para $\overline{A}$ y 1 para A.

Es importante señalar que el mapa de Karnaugh es una representación gráfica de una tabla de verdad y que, por tanto, existe una asociación unívoca entre ambas. La tabla de verdad tiene una fila por cada mintérmino, mientras que el mapa de Karnaugh tiene un cuadrado por mintérmino. Esto se ilustra en la figura 3.1g. De manera análoga, también existe una correspondencia unívoca entre las filas de la tabla de verdad y los cuadrados del mapa de Karnaugh si se utilizan los maxtérminos.

La figura 3.2 muestra el desarrollo del mapa de Karnaugh para tres variables. Un punto importante que requiere un análisis cuidadoso es el paso entre las figuras 3.2c y d. Por ejemplo, consideremos el mintérmino m_0. Este mintérmino es adyacente a m_1, m_2 y m_4 en la figura 3.2c. Sin embargo, en la figura 3.2d, m_0 no está físicamente adyacente a m_4. Para salvar esta inconsistencia, consideramos los extremos izquierdo y derecho del mapa como *la misma línea*. En otras palabras, podemos doblar el extremo izquierdo hasta que toque al extremo derecho, lo que hace al mapa de Karnaugh para tres variables semejar a un cilindro. En la práctica, graficamos el mapa como en la figura 3.2e o f, e imaginamos que los extremos izquierdo y derecho coinciden.

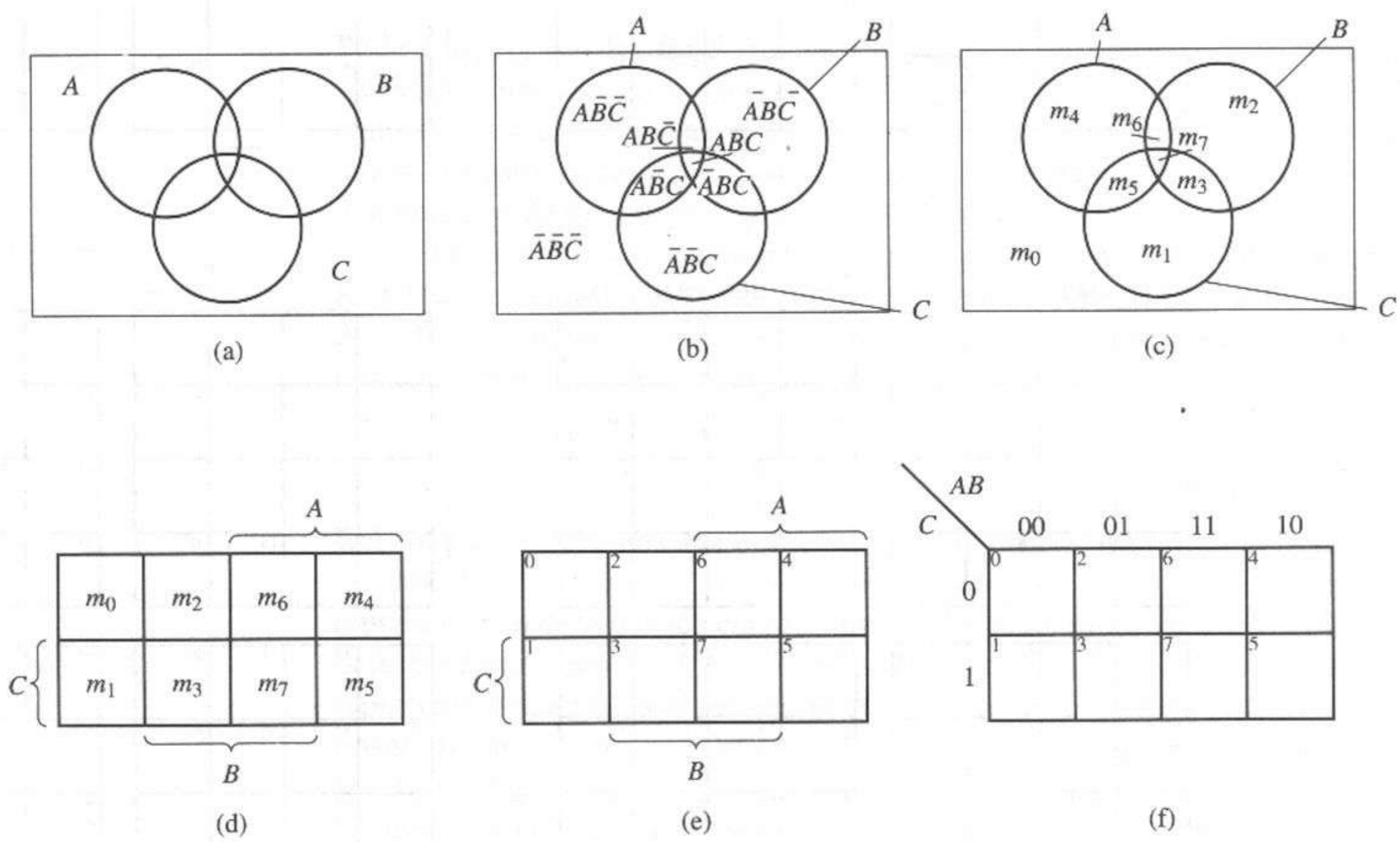


Figura 3.2 Diagrama de Venn y mapa de Karnaugh equivalente para tres variables.

3.3.2 Mapas K de cuatro o más variables

Las figuras 3.3a a f muestran los mapas K para cuatro, cinco y seis variables. Observe que el mapa de cuatro variables es sólo una extensión del mapa de tres

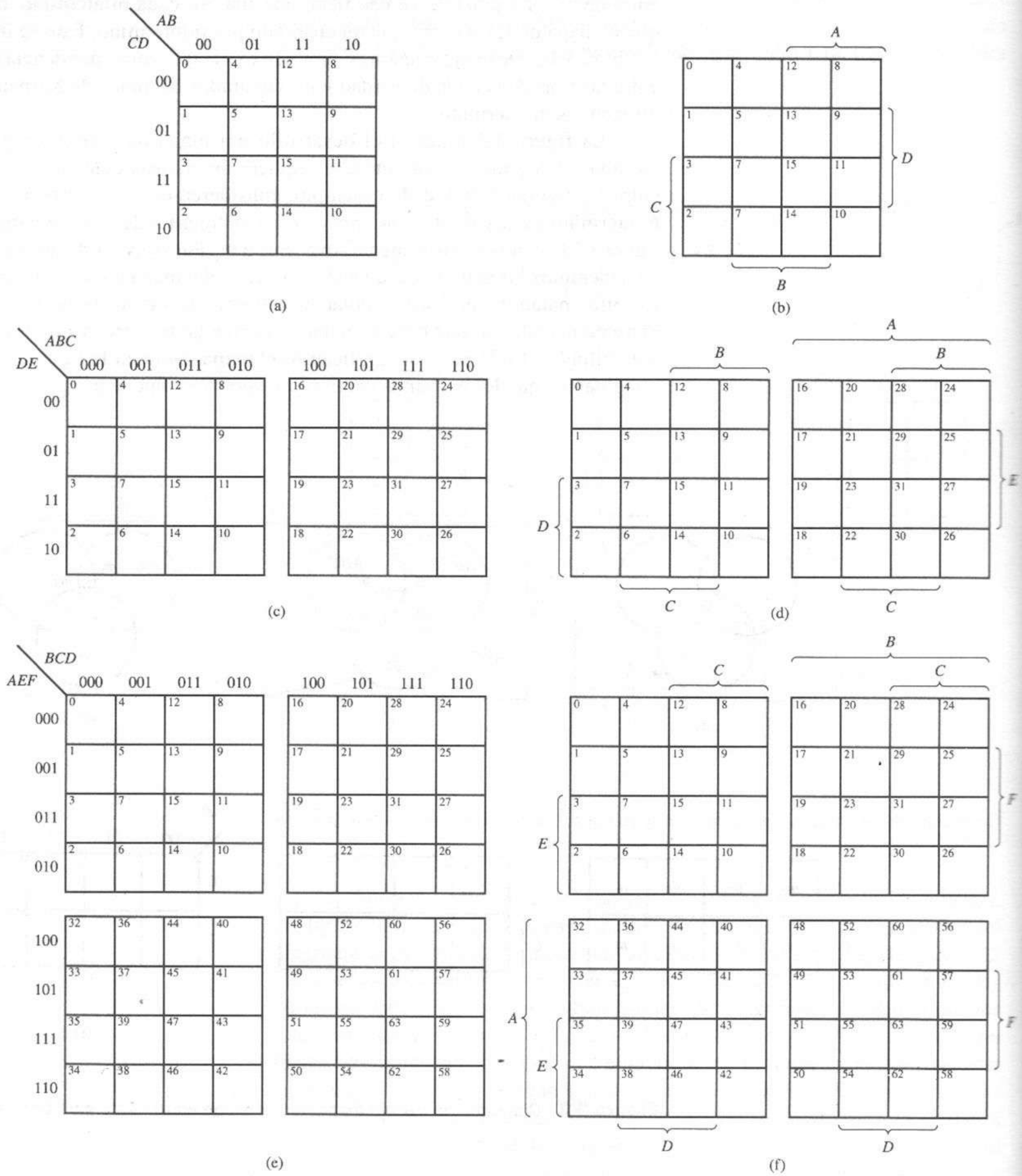


Figura 3.3 Mapas K para cuatro, cinco y seis variables. **(a)** $f(A, B, C, D)$, forma 1. **(b)** $f(A, B, C, D)$, forma 2. **(c)** $f(A, B, C, D, E)$, forma 1. **(d)** $f(A, B, C, D, E)$, forma 2. **(e)** $f(A, B, C, D, E, F)$, forma 1. **(f)** $f(A, B, C, D, E, F)$, forma 2.

variables. El mapa de cinco variables para $f(A, B, C, D, E)$ se separa en dos mitades: la mitad izquierda representa los mintérminos que contienen $\overline{A}$ y la derecha representa los mintérminos que contienen A. Hay que pensar que las dos mitades están una encima de la otra, de modo que las celdas adyacentes en sentido vertical sólo difieran en la variable A y, por tanto, son adyacentes. Por ejemplo, las celdas correspondientes a los mintérminos m_5 $(\overline{A}\overline{B}C\overline{D}E)$ y m_{21} $(A\overline{B}C\overline{D}E)$ son adyacentes.

De manera análoga, dividimos el mapa de seis variables en cuatro cuadrantes, cada uno de los cuales representa una combinación de las variables A y B. Imaginamos a los cuadrantes apilados uno encima del otro, de modo que las celdas adyacentes en sentido vertical son adyacentes.

Los mapas presentados en las figuras 3.1e y f, 3.2e y f, y 3.3a a f combinan todas las características familiares que los diseñadores lógicos usan en la síntesis de circuitos de conmutación. Podemos utilizar cualquiera de los dos formatos para un mapa K de n variables, según las preferencias personales. Los mapas K de más de seis variables no son prácticos para la mayor parte de los problemas.

3.4 Trazo de funciones en forma canónica sobre el mapa K

Podemos expresar las funciones de conmutación de varias formas, desde las listas de mintérminos o maxtérminos hasta las expresiones sencillas SOP/POS o expresiones más complejas. Sin embargo, cada una tiene una forma POS/SOP única. En esta sección veremos algunos métodos para graficar funciones de conmutación de diferentes formas sobre un mapa de Karnaugh.

Podemos graficar fácilmente las funciones de conmutación sobre un mapa K si éstas se expresan en forma canónica, pues cada mintérmino o maxtérmino de la forma canónica corresponde a una celda sobre el mapa K. Supongamos que queremos determinar el mapa K para la siguiente función:

$$\begin{aligned} f(A, B, C) &= m(0, 3, 5) = m_0 + m_3 + m_5 \\ &= \prod M(1, 2, 4, 6, 7) = M_1 M_2 M_4 M_6 M_7 \end{aligned}$$

Del capítulo 2, sabemos que es fácil deducir la lista de maxtérminos a partir de la lista de mintérminos, y viceversa. En primer lugar, consideremos la representación de la función como la suma de los mintérminos 0, 3 y 5. Utilizamos la forma de diagrama de Venn del mapa K, y tenemos que la función $f(A, B, C)$ representa las áreas sombreadas de la figura 3.4a. La figura 3.4b muestra la misma función sobre un mapa K. Observe que, por lo general, no se utilizan áreas sombreadas en los mapas K, sino que se emplean los unos y ceros de las tablas de verdad; cada área sombreada (cada mintérmino) se representa con 1, y cada área no sombreada (cada maxtérmino), por 0. En estas condiciones, el mapa K de la figura 3.4b corresponde de manera directa a la tabla de verdad de la función, de modo que cada celda del mapa K corresponde a una fila de la tabla de verdad. Cuando representamos la función mediante una suma de mintérminos, normalmente omitimos los maxtérminos del mapa y representamos la función como en la figura 3.4c. De manera análoga, si la función se expresa como un producto de maxtérminos, omitimos los mintérminos y representamos la función como en la figura 3.4d.

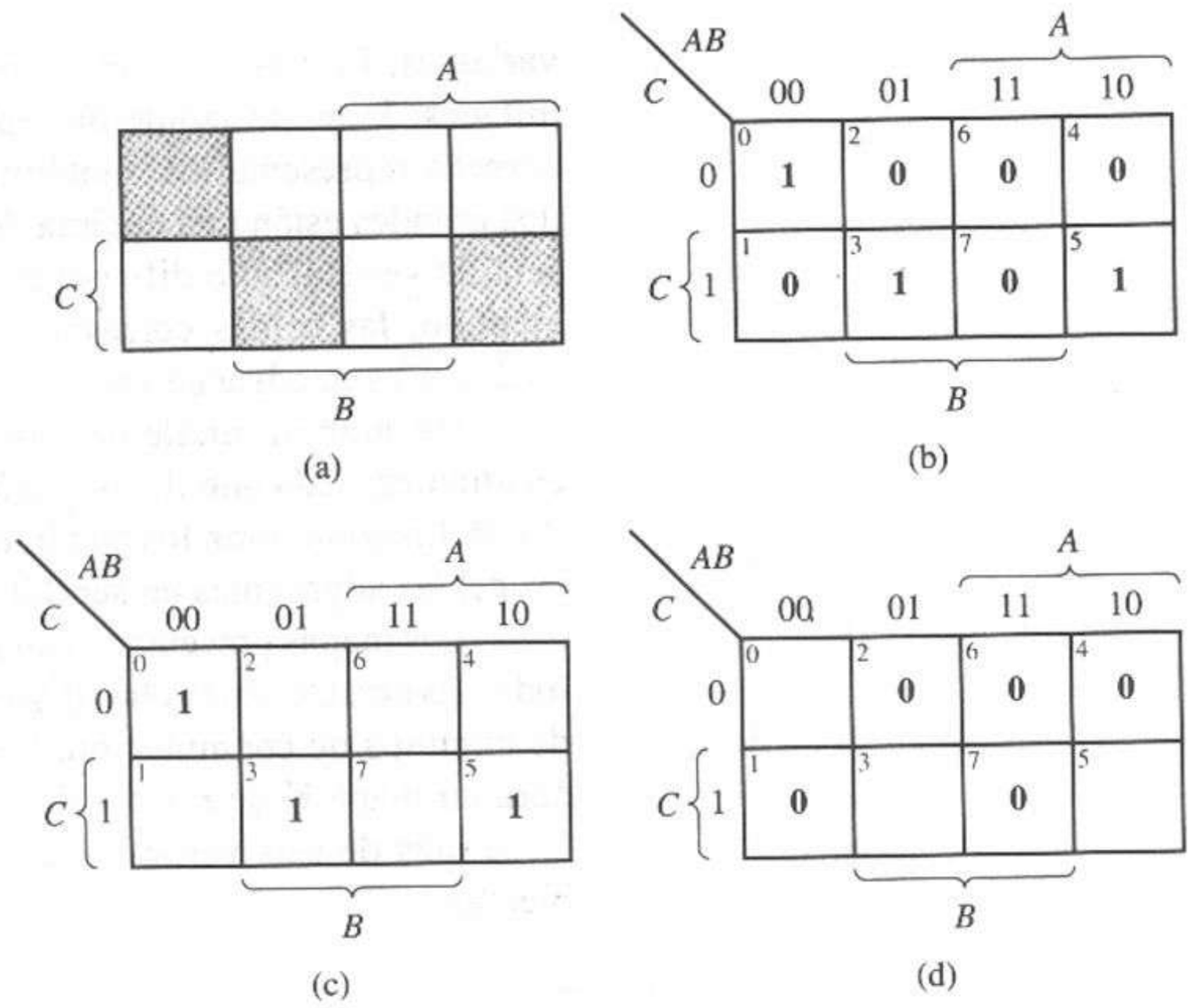


Figura 3.4 Graficación de funciones sobre mapas K. **(a)** Forma de diagrama de Venn. **(b)** Los unos representan los mintérminos y los ceros representan los maxtérminos. **(c)** Graficación sólo de los mintérminos. **(d)** Graficación sólo de los maxtérminos.

EJEMPLO 3.4

Graficar la siguiente función sobre un mapa K:

$$f(a,b,Q,G) = m(0,3,5,7,10,11,12,13,14,15)$$
$$= \prod M(1,2,4,6,8,9)$$

La función, expresada como suma de mintérminos, se grafica en la figura 3.5a sobre una versión del mapa K rotulada según ambas convenciones. La función, expresada como un producto de maxtérminos, aparece en el mapa K de la figura 3.5b.

Un punto muy importante es el orden de las variables. Como mostramos en la última sección, la cantidad de mintérminos y maxtérminos de la lista cambia si se altera el orden de las variables. Por tanto, el orden de las variables en la función fija el orden de las variables en el mapa K.

EJEMPLO 3.5

Repetir el ejemplo 3.4 con las variables reordenadas para obtener $f(Q, G, b, a)$.

Primero escribimos los mintérminos de $f(a, b, Q, G)$.

$$f(a,b,Q,G) = \sum m(0,3,5,7,10,11,12,13,14,15)$$
$$= \bar{a}\bar{b}\bar{Q}\bar{G} + \bar{a}\bar{b}QG + \bar{a}b\bar{Q}G + \bar{a}bQG + a\bar{b}Q\bar{G}$$
$$+ a\bar{b}QG + ab\bar{Q}\bar{G} + ab\bar{Q}G + abQ\bar{G} + abQG$$

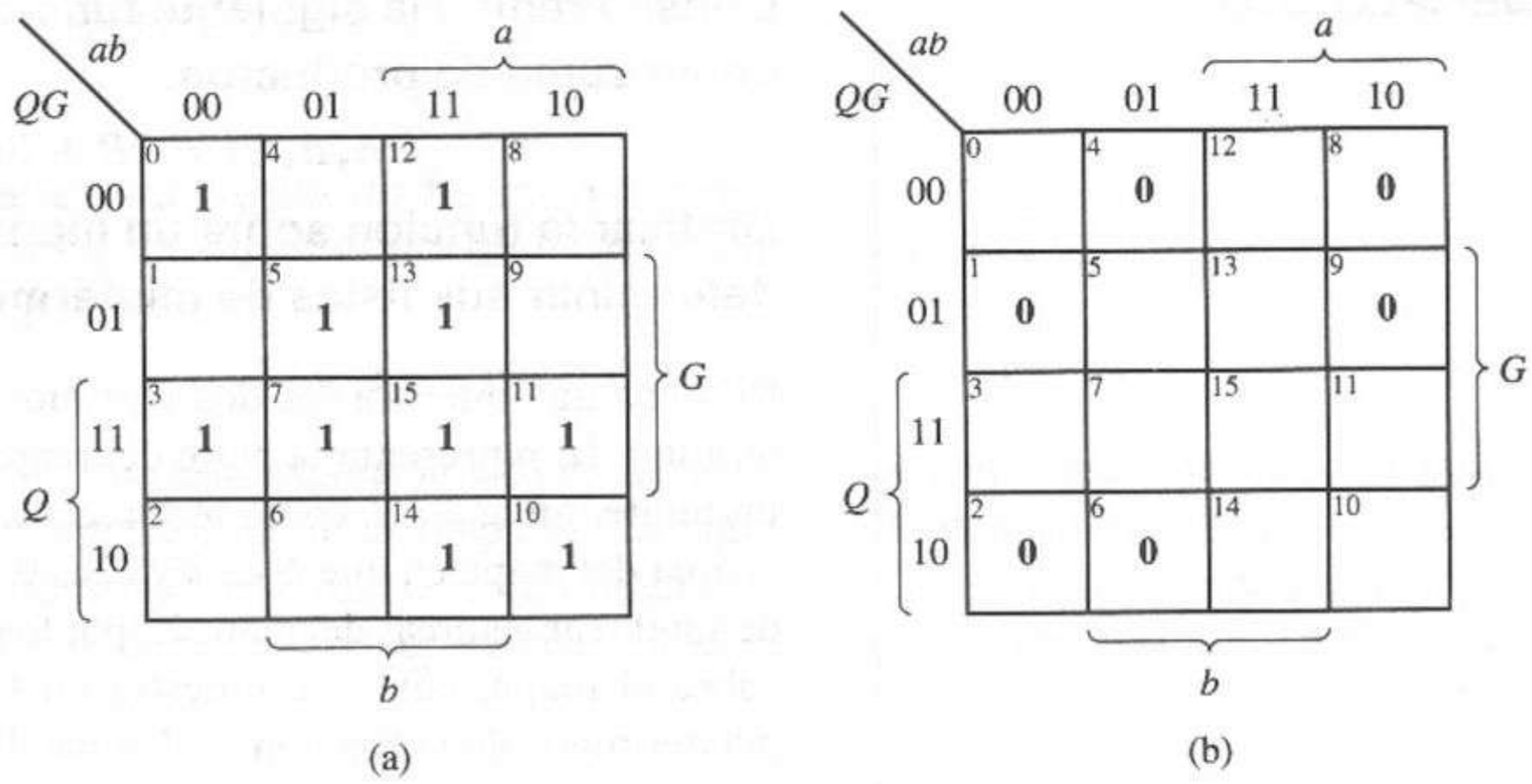


Figura 3.5 Mapas K para $f(a, b, Q, G)$ del ejemplo 3.4. **(a)** Forma con mintérminos. **(b)** Forma con maxtérminos.

A continuación, reordenamos las variables:

$$\begin{aligned} f(Q, G, b, a) &= \bar{Q}\bar{G}\bar{b}\bar{a} + QG\bar{b}\bar{a} + \bar{Q}Gb\bar{a} + QGb\bar{a} + Q\bar{G}\bar{b}a \\ &\quad + QG\bar{b}a + \bar{Q}\bar{G}ba + \bar{Q}Gba + Q\bar{G}ba + QGba \\ &= \sum m(0, 12, 6, 14, 9, 13, 3, 7, 11, 15) \\ &= \sum m(0, 3, 6, 7, 9, 11, 12, 13, 14, 15) \end{aligned}$$

Graficamos la función sobre el mapa de la figura 3.6, que es equivalente al de la figura 3.5a.

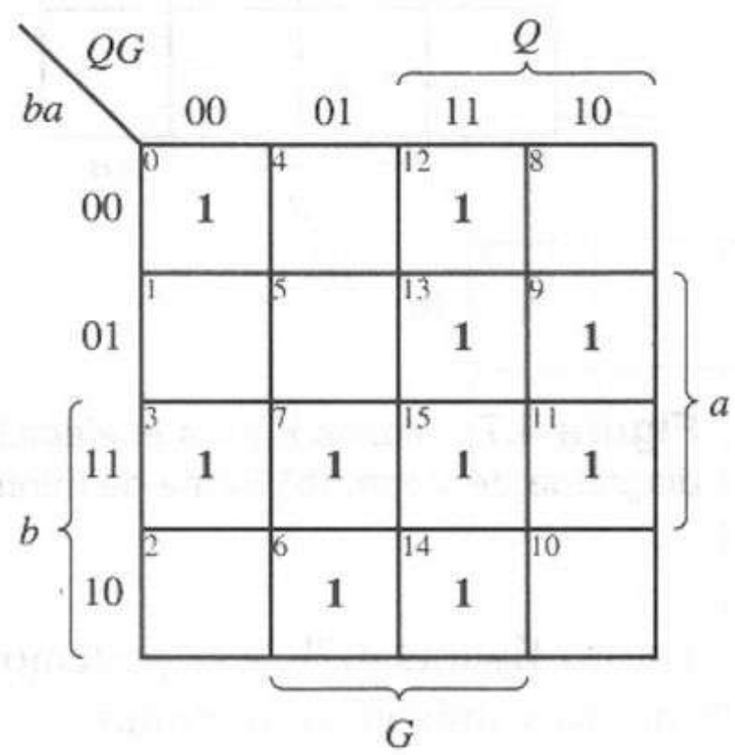


Figura 3.6 Mapa K de la figura 3.5(a) con las variables reordenadas: $f(Q, G, b, a)$.

Los mapas K también pueden servir para desarrollar una función en forma canónica. A fin de ilustrar esta técnica, continuaremos utilizando una combinación de las dos formas de mapas K de la figura 3.3.

EJEMPLO 3.6

Consideremos la siguiente función, expresada como suma de productos.

$$f(A,B,C) = AB + BC$$

Graficar la función sobre un mapa K y determinar sus listas de mintérminos y maxtérminos.

El mapa que muestra los dos términos producto aparece en la figura 3.7a. El término AB representa la parte del mapa en que A y B son iguales a 1, es decir, los mintérminos 6 y 7, como se aprecia en la figura 3.7b. El término $B\overline{C}$ representa el área del mapa en que B es 1 y C es 0, es decir, los mintérminos 2 y 6. En vez de sombrear las áreas del mapa K, por lo general colocamos los unos directamente sobre el mapa, como se muestra en la figura 3.7b. El mapa que ilustra los maxtérminos de la función se deduce directamente del mapa de mintérminos y aparece en la figura 3.7c.

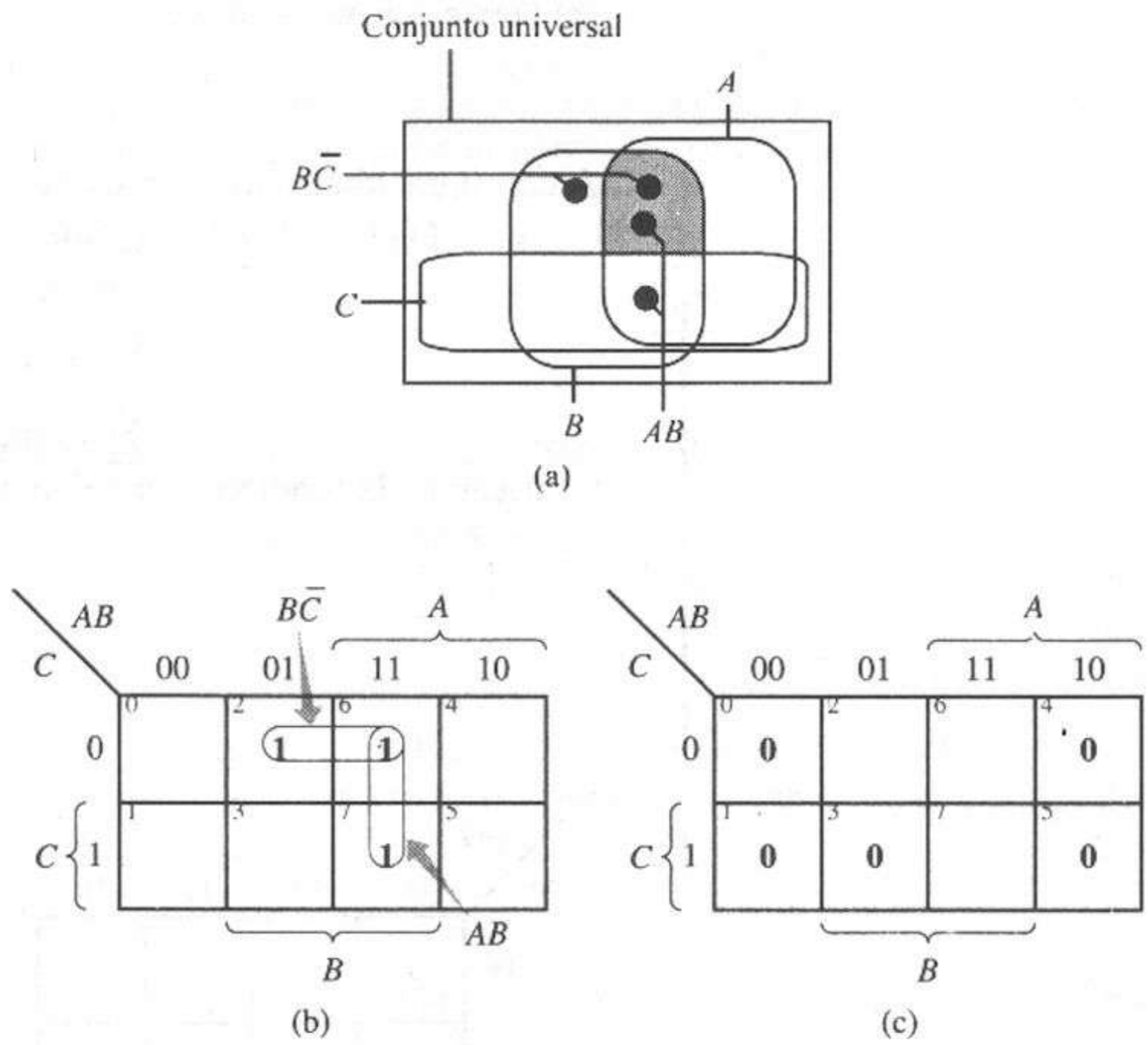


Figura 3.7 Mapas K para el ejemplo 3.6. **(a)** Mapa trazado en forma de diagrama de Venn. **(b)** Suma de mintérminos. **(c)** Maxtérminos de la función.

De las figuras 3.7b y c, podemos expresar la función en las formas de mintérminos y maxtérminos como

$$f(A, B, C) = m(2, 6, 7) = \prod M(0, 1, 3, 4, 5)$$

En este ejemplo, observamos que el mintérmino 6 se usa dos veces, es decir, es "cubierto" por los dos términos producto originales. El uso múltiple de los mintérminos y maxtérminos es con frecuencia la regla en las funciones de

conmutación, más que la excepción. Además, observemos que en realidad no era necesario trazar el mapa K de la figura 3.7c, pues podemos identificar los maxtérminos en forma directa sobre el mapa K de la figura 3.7b, tomando nota de las celdas que no son iguales a 1.

EJEMPLO 3.7

Graficar la siguiente función sobre el mapa K y determinar sus listas de mintérminos y maxtérminos.

$$f(A,B,C,D) = (A + C)(B + C)(\bar{B} + \bar{C} + D)$$

Puesto que esta expresión está en forma POS, podemos graficar los ceros (sus maxtérminos) sobre el mapa K. La figura 3.8a es el mapa que muestra los maxtérminos. El término $A + C$ obliga a la función a asumir el valor 0 cuando $A = C = 0$, de modo que representa el área del mapa para la cual $A = 0$ y $C = 0$, y

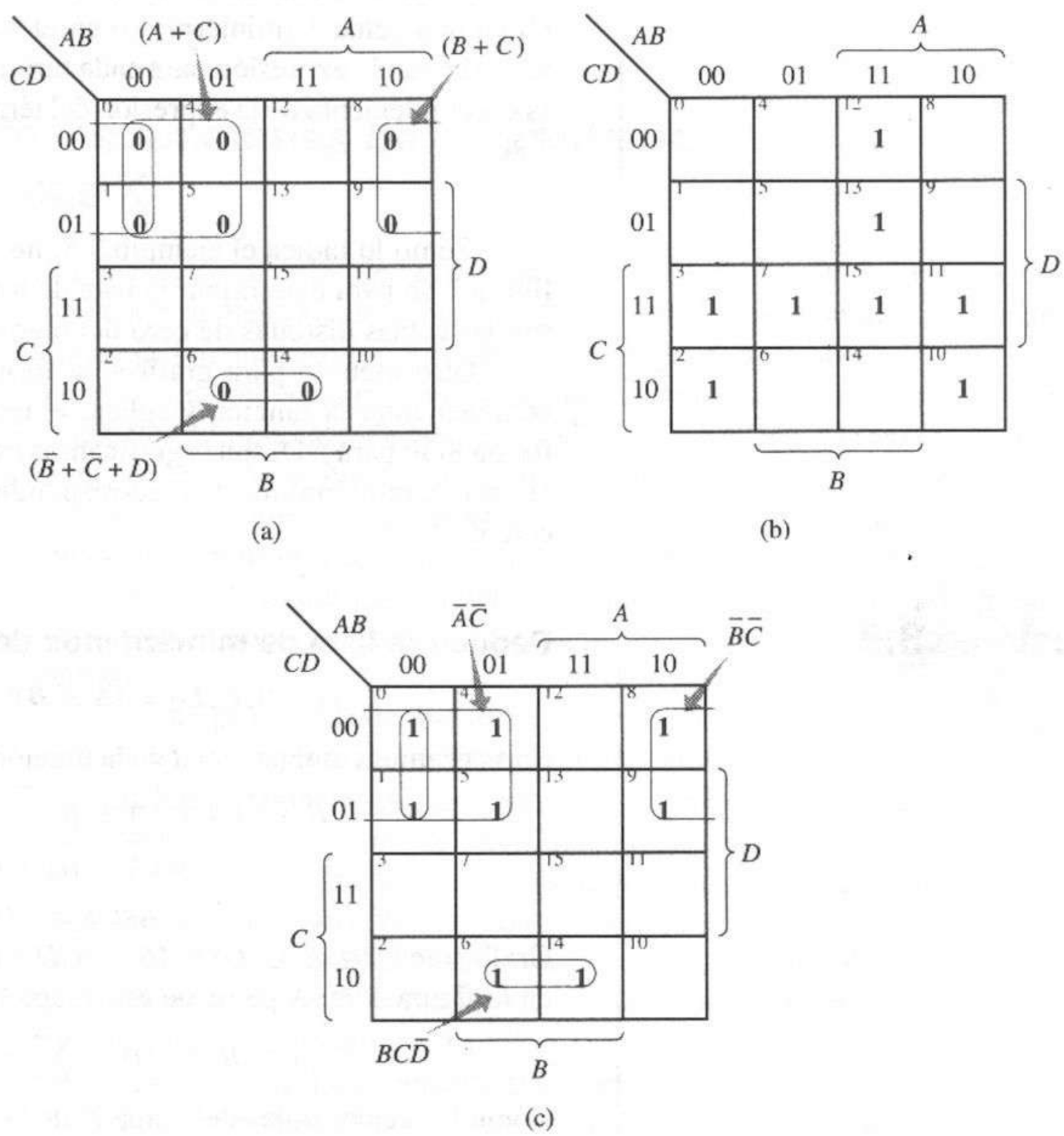


Figura 3.8 Mapas K para el ejemplo 3.7. **(a)** Mapa K que muestra los maxtérminos. **(b)** Mapa K que muestra los mintérminos. **(c)** Mapa K que muestra los mintérminos de $\bar{f}(A, B, C, D)$.

esto representa, es decir, abarca, los maxtérminos 0, 1, 4 y 5. De manera análoga, $B + C$ representa los maxtérminos 0, 1, 8 y 9. El término $\overline{B} + \overline{C} + D$ representa los maxtérminos 6 y 14, pues el término es 0 cuando $B = 1$, $C = 1$ y $D = 0$. La figura 3.8b muestra el mapa correspondiente, que ilustra los mintérminos.

De las figuras 3.8a y b, podemos expresar la función como

$$f(A, B, C, D) = \prod M(0, 1, 4, 5, 6, 8, 9, 14)$$
$$= \sum m(2, 3, 7, 10, 11, 12, 13, 15)$$

Algunos diseñadores encuentran difícil la graficación de las expresiones POS. Un método alternativo consiste en complementar la función y aplicar el teorema de DeMorgan para obtener una expresión SOP para $\overline{f}(A, B, C, D)$.

$$\bar{f}(A, B, C, D) = \overline{(A + C)(B + C)(\bar{B} + \bar{C} + D)}$$
$$= \overline{(A + C)} + \overline{(B + C)} + \overline{(\bar{B} + \bar{C} + D)}$$
$$= \bar{A}\bar{C} + \bar{B}\bar{C} + BC\bar{D}$$

Graficamos entonces la forma SOP de $\overline{f}(A, B, C, D)$ sobre un mapa K como en la figura 3.8c. Recordemos que los mintérminos de $\overline{f}(A, B, C, D)$ son los maxtérminos de $f(A, B, C, D)$ y viceversa, por lo que obtenemos el mapa K de la figura 3.8a convirtiendo cada celda 0 (maxtérmino) en el mapa K de $\overline{f}(A, B, C, D)$ en una celda 1 (mintérmino) en el mapa K de $f(A, B, C, D)$. Observemos también que la expresión para cada término producto indicado en la figura 3.8b es el complemento de la expresión del término suma correspondiente de la figura 3.8c.

Como lo indica el ejemplo 3.5, no era necesario graficar el mapa K de la figura 3.8b para determinar la lista de mintérminos, pues los mintérminos sólo son las celdas distintas de cero del mapa K de la figura 3.8a.

Otro método para graficar la forma POS de una función f consiste en complementar la función y aplicar el teorema de DeMorgan para obtener una forma SOP para $\overline{f}$. Después graficamos $\overline{f}$ sobre el mapa K, del cual podemos leer la lista de mintérminos de $\overline{f}$, correspondiendo los mintérminos de f a las celdas cero de $\overline{f}$.

EJEMPLO 3.8

Deducir la lista de mintérminos de la función

$$f(A, B, C, D) = (\bar{A} + \bar{B})(\bar{A} + C + \bar{D})(\bar{B} + \bar{C} + \bar{D})$$

Comenzamos complementando la función y aplicamos el teorema de DeMorgan:

$$\bar{f}(A, B, C, D) = \overline{(\bar{A} + \bar{B})(\bar{A} + C + \bar{D})(\bar{B} + \bar{C} + \bar{D})}$$
$$= \overline{(\bar{A} + \bar{B})} + \overline{(\bar{A} + C + \bar{D})} + \overline{(\bar{B} + \bar{C} + \bar{D})}$$
$$= AB + A\bar{C}D + BCD$$

Graficamos $\overline{f}(A, B, C, D) = AB + A\overline{C}D + BCD$ sobre el mapa K como se muestra en la figura 3.9a. A partir de este mapa K, podemos escribir

$$\bar{f}(A, B, C, D) = \sum m(7, 9, 12, 13, 14, 15)$$

Como las celdas nulas del mapa K de la figura 3.9a representan $f(A, B, C, D)$, una inspección del mapa K nos permite escribir

$$f(A, B, C, D) = \sum m(0, 1, 2, 3, 4, 5, 6, 8, 10, 11)$$

Graficamos la función sobre el mapa K de la figura 3.9b.

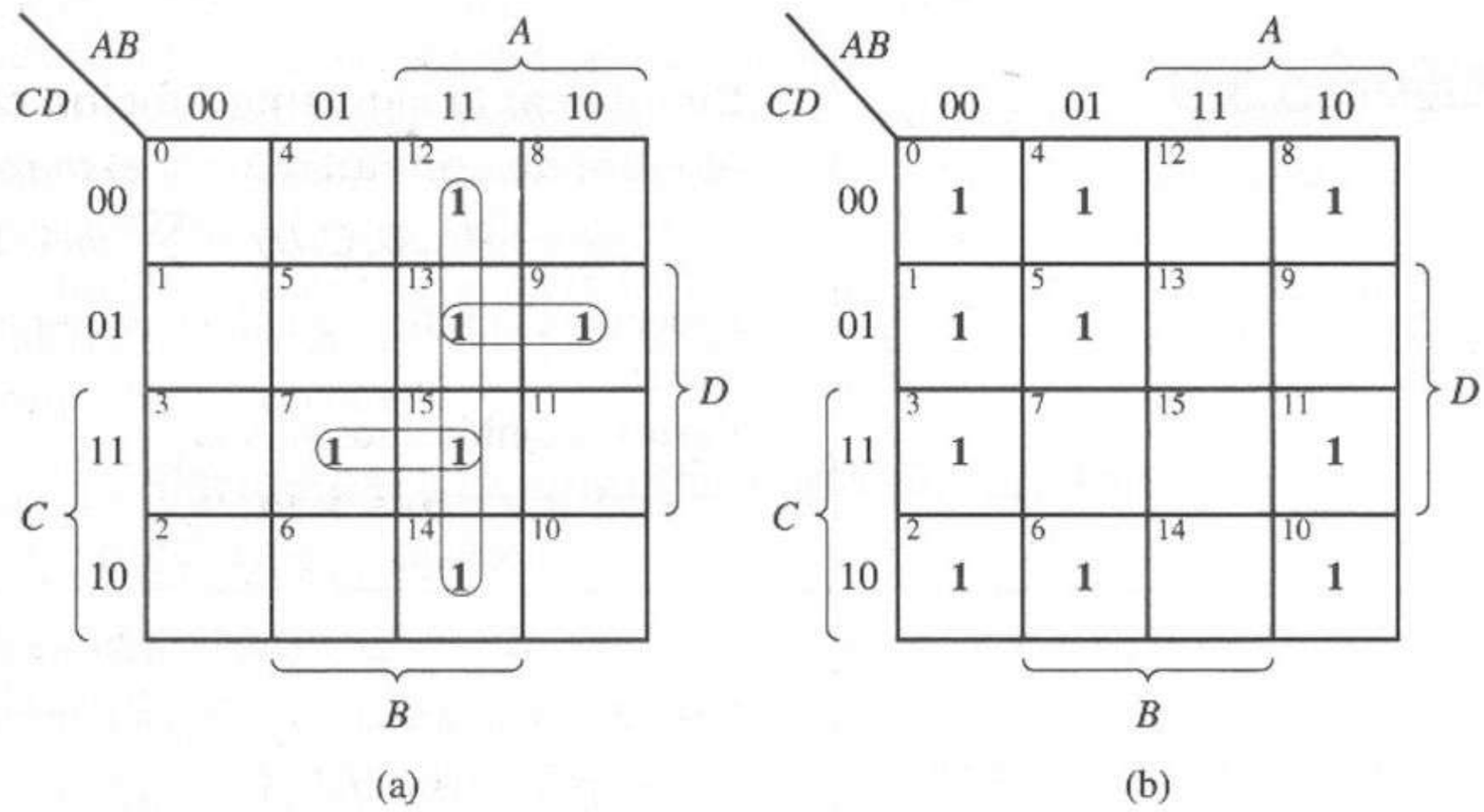


Figura 3.9 Mapas K para el ejemplo 3.8. **(a)** Mapa K de $\overline{f}(A, B, C, D)$. **(b)** Mapa K correspondiente de $f(A, B, C, D)$.

3.5 Simplificación de las funciones de conmutación mediante los mapas K

En la sección anterior dedujimos una expresión SOP o POS para una función mediante su mapa K, sin considerar si ésta era la expresión más simple posible para esa función. Ahora utilizaremos el mapa K para obtener una expresión de suma de productos mínima para una función de conmutación. Por suma de productos mínima entendemos una expresión equivalente a la expresión original, pero que contiene un número mínimo de términos producto donde están presentes un número mínimo de literales. La minimización del número de términos producto permite utilizar el menor número de compuertas para realizar la función, mientras que la minimización del número de literales permite usar compuertas con el *fan-in* mínimo posible (y, por tanto, el costo mínimo) para realizar cada término.

La simplificación de funciones sobre el mapa K se facilita por el hecho de que, sobre el mapa, los términos de conmutación adyacentes desde el punto de vista lógico también son adyacentes físicamente. Definimos los mintérminos *adyacentes desde el punto de vista lógico* como sigue: dos mintérminos, m_i y m_j, son adyacentes lógicamente si difieren sólo en una posición de variable. Por ejemplo, $AB\overline{C}\overline{D}$ y $AB\overline{C}D$ son mintérminos de cuatro variables adyacentes lógicamente, ya que sólo difieren en la posición de la variable D. Por el teorema 6(a) sabemos que $AB\overline{C}\overline{D} + AB\overline{C}D = AB\overline{C}$; por tanto, los términos $AB\overline{C}\overline{D}$ y $AB\overline{C}D$ se *combinan*, eliminando la variable D. En general, podemos combinar dos términos adyacentes lógicamente eliminando una variable.

En el mapa K indicamos los términos que se pueden combinar trazando un anillo alrededor de los términos que, al combinarse, den una expresión más sencilla, es decir, con menos literales.

El siguiente ejemplo ilustra el proceso de combinación de términos adyacentes lógicamente, utilizando el álgebra de conmutación y los métodos del mapa K.

EJEMPLO 3.9

Simplificar la siguiente función con el álgebra de conmutación y el mapa K.

$$f(A,B,C,D) = \sum m(1,2,4,6,9)$$

Podemos simplificar esta función por medio del álgebra de conmutación como sigue:

Paso 1. Combinamos m_1 y m_9.

$$\begin{aligned} f(A,B,C,D) &= \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D \\ &= (\bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D) + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} \\ &= \bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} \end{aligned}$$

Paso 2. Combinamos m_2 y m_6, duplicando primero m_6.

$$\begin{aligned} f(A,B,C,D) &= \bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + (\bar{A}BC\bar{D} + \bar{A}BC\bar{D}) \\ &= \bar{B}\bar{C}D + (\bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}) + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} \\ &= \bar{B}\bar{C}D + \bar{A}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} \end{aligned}$$

Paso 3. Combinamos m_4 y m_6.

$$\begin{aligned} f(A,B,C,D) &= \bar{B}\bar{C}D + \bar{A}C\bar{D} + (\bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D}) \\ &= \bar{B}\bar{C}D + \bar{A}C\bar{D} + \bar{A}B\bar{D} \end{aligned}$$

La figura 3.10 muestra la simplificación correspondiente con el mapa K. La simplificación implica encerrar en un círculo los conjuntos de cuadrados adyacentes físicamente, correspondientes a los grupos de mintérminos adyacentes lógicamente. Observe que, como las orillas opuestas del mapa coinciden, las formas de herradura de la figura 3.10 son en realidad círculos. Cada círculo indica la manera en que los mintérminos encerrados en el círculo se combinan para obtener una expresión de conmutación más sencilla.

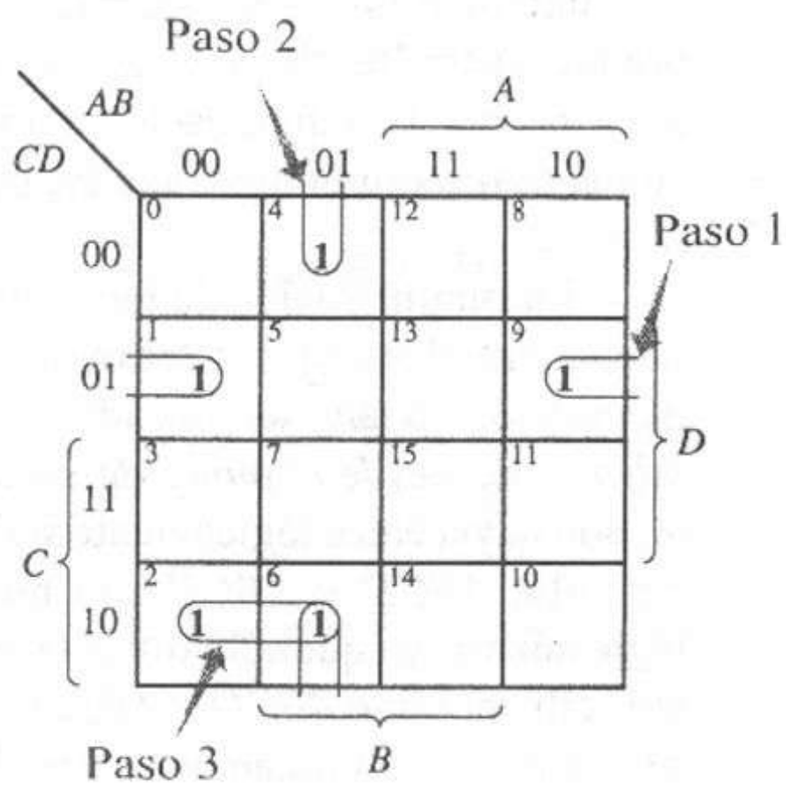


Figura 3.10 Mapa K para el ejemplo 3.9.

En el paso 1 combinamos los mintérminos m_1 y m_9, adyacentes en el mapa K, encerrando en un círculo los cuadrados correspondientes. Al comparar estos dos cuadrados, vemos que la variable que cambia de 0 en m_1 a 1 en m_9 es A; por tanto, al combinar los dos mintérminos, esta variable se elimina, como se muestra

en el paso 1 del método de álgebra de conmutación. Combinamos los mintérminos m_4 y m_6 sobre el mapa K en el paso 2. Al comparar estos dos cuadrados vemos que sólo difieren en la variable C; por tanto, eliminamos esta variable al combinar los cuadrados 4 y 6. Por ultimo, en el paso 3, la combinación de los cuadrados 2 y 6 elimina la variable B. Así, los tres pasos en el mapa K son equivalentes a los pasos correspondientes indicados en la simplificación mediante álgebra de conmutación. Como se aprecia en los pasos 2 y 3, podemos utilizar los mintérminos más de una vez, ya que, como recordará el lector, $X = X + X$ por idempotencia (teorema 1).

3.5.1 Criterios para la simplificación de funciones mediante mapas K

Hay cinco puntos importantes que debemos recordar al simplificar funciones sobre mapas K:

1. Cada cuadrado (mintérmino) sobre un mapa K de dos variables tiene dos cuadrados (mintérminos) adyacentes lógicamente; cada cuadrado sobre un mapa K de tres variables tiene tres cuadrados adyacentes, etcétera. En general, cada cuadrado en un mapa K de n variables tiene n cuadrados adyacentes lógicamente, de modo que cada par de cuadrados adyacentes difiere precisamente en una variable.
2. Al combinar los términos (cuadrados) en un mapa K, siempre agrupamos cuadrados en potencias de 2; es decir, dos cuadrados, cuatro cuadrados, ocho cuadrados, etc. Al agrupar dos cuadrados eliminamos una variable, al agrupar cuatro cuadrados eliminamos dos variables, etc. En general, al agrupar 2^n cuadrados eliminamos n variables.
3. Debemos agrupar tantos cuadrados como sea posible; cuanto mayor sea el grupo, habrá un número menor de literales en el término producto resultante.
4. Debemos formar el menor número posible de grupos que cubran todos los cuadrados (mintérminos) de la función. Un mintérmino está *cubierto* si está incluido al menos en un grupo. Si hay menos grupos, será menor el número de términos producto en la función minimizada. Podemos utilizar cada mintérmino cuantas veces sea necesario en los pasos 4 y 5; sin embargo, debemos utilizarlo al menos una vez. Tan pronto hayamos utilizado todos los mintérminos al menos una vez, nos detenemos. Un mintérmino utilizado al menos en un grupo ha sido *cubierto*.
5. Al combinar cuadrados en el mapa, hay que comenzar siempre por los cuadrados donde existe el menor número de cuadrados adyacentes (los cuadrados "más solitarios" en el mapa). Los mintérminos con varios mintérminos adyacentes (llamados *adyacencias*) ofrecen más combinaciones posibles y, por tanto, deben combinarse más adelante en el proceso de minimización.

3.5.2 Terminología general para la minimización de una función de conmutación

En el análisis anterior ilustramos la relación entre los procedimientos del álgebra de conmutación y de los mapas K para simplificar una función de conmutación. Ahora definiremos cuatro términos que no sólo son útiles en la simplificación

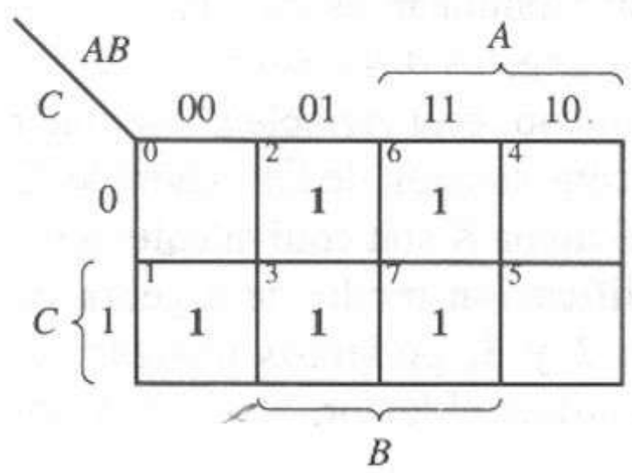


Figura 3.11 Mapa K que muestra los implicantes.

con mapas K, sino que también proporcionan una base para otras técnicas más generales de minimización de funciones de conmutación que presentaremos posteriormente. Estos términos son *implicante*, *implicante primo*, *implicante primo esencial* y *cubierta*.

Un *implicante* es un término producto (es decir, un producto de una o más literales) que puede servir para cubrir mintérminos de la función. En el mapa K de la figura 3.11, hay 11 implicantes:

5 mintérminos: $\{\overline{A}\overline{B}C, \overline{A}B\overline{C}, \overline{A}BC, AB\overline{C}, ABC\}$

5 grupos de dos mintérminos: $\{\overline{A}B, AB, \overline{A}C, B\overline{C}, BC\}$

1 grupo de cuatro mintérminos: $\{B\}$

Un *implicante primo* es un implicante que no es parte de (no está cubierto por) algún otro implicante de la función. Del ejemplo anterior, recuerde que al combinar mintérminos (implicantes) eliminamos variables. Al combinar implicantes (en potencias de 2) en grupos maximales, formamos implicantes primos. En el mapa K, un implicante primo equivale a un conjunto de cuadrados que no es subconjunto de algún conjunto con un número mayor de cuadrados. Los implicantes primos representan los máximos agrupamientos de mintérminos deducibles para la función. En el mapa K de la figura 3.11 sólo hay dos implicantes primos: B y $\overline{A}C$. El implicante primo B cubre los implicantes $\overline{A}\overline{B}\overline{C}$, $\overline{A}BC$, $AB\overline{C}$, ABC, $\overline{A}B$, AB, $B\overline{C}$ y BC. El implicante primo $\overline{A}C$ cubre los implicantes $\overline{A}\overline{B}C$ y $\overline{A}BC$.

Un *implicante primo esencial* es un implicante primo que cubre al menos un mintérmino que no está cubierto por algún otro implicante primo. En el mapa K de la figura 3.11, el implicante primo $\overline{A}C$ es esencial, pues es el único implicante primo que cubre el mintérmino 1, y el implicante primo B es esencial porque es el único implicante primo que cubre los mintérminos 2, 6 y 7. Podemos identificar con facilidad un implicante primo esencial en el mapa K, observando que cubre al menos un mintérmino encerrado en un círculo sólo una vez.

Por último, una *cubierta* de una función es un conjunto de implicantes primos tal que todos los mintérminos de la función están contenidos en (cubiertos por) al menos un implicante primo. Debemos seleccionar todos los implicantes primos esenciales de una función en la cubierta de una función. Para el mapa K de la figura 3.11, el conjunto de implicantes primos $\{B, \overline{A}C\}$ representa una cubierta de la función.

3.5.3 Algoritmos para la deducción de formas SOP minimales a partir de mapas K

Nuestro objetivo principal al minimizar una función es determinar un conjunto mínimo de implicantes primos que cubran una función, del cual se podría deducir una expresión mínima de suma de productos. Con esto en mente, presentamos ahora dos algoritmos para determinar la cubierta mínima de una función indicada sobre un mapa K. Estos algoritmos están diseñados para ayudar al usuario a seguir los cinco criterios presentados con anterioridad, y con ello simplificar el proceso de minimización y contar con una alta probabilidad de determinar la cubierta mínima de cualquier función arbitraria.

Algoritmo 3.1

1. Contar el número de adyacencias para cada mintérmino en el mapa K.
2. Seleccionar un mintérmino no cubierto con el menor número de adyacencias. Éste se elige de manera arbitraria si existen varias opciones.
3. Generar un implicante primo para este mintérmino e incluirlo en la cubierta. Si este mintérmino está cubierto por más de un implicante primo, se selecciona el que cubre más mintérminos no cubiertos.
4. Repetir los pasos 2 y 3 hasta cubrir todos los mintérminos.

Este algoritmo es fácil de aplicar y generalmente determina una solución mínima. Sin embargo, como se permiten elecciones arbitrarias en los pasos 1 y 2, no es óptimo. Por tanto, no garantiza determinar la cubierta mínima para toda función.

El siguiente algoritmo para generar una cubierta mínima de una función a partir de un mapa K es una alternativa eficiente al algoritmo 3.1 si el número de implicantes primos no es demasiado grande.

Algoritmo 3.2

1. Encerrar en un círculo todos los implicantes primos en el mapa K.
2. Identificar y seleccionar todos los implicantes primos esenciales para la cubierta.
3. Seleccionar un subconjunto mínimo de los implicantes primos restantes para completar la cubierta; es decir, cubrir aquellos mintérminos no cubiertos por los implicantes primos esenciales.

Este algoritmo genera más términos y, por tanto, requiere más trabajo que el algoritmo 3.1. Sin embargo, como se identifican y consideran todos los implicantes primos, por lo general es mejor para determinar una cubierta mínima. De nuevo, hay que tomar decisiones arbitrarias en el paso 3, por lo que el algoritmo no es óptimo. Una vez que tenemos una cubierta mínima de la función, deducimos la forma mínima SOP escribiendo la suma de los implicantes primos en la cubierta.

Ahora ilustraremos la aplicación de estos dos algoritmos con los siguientes ejemplos.

EJEMPLO 3.10

Utilizar el mapa K y el algoritmo 3.1 para simplificar la siguiente función.

$$f(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$$

Para iniciar el proceso de simplificación, graficamos primero la función sobre el mapa, como se muestra en la figura 3.12a.

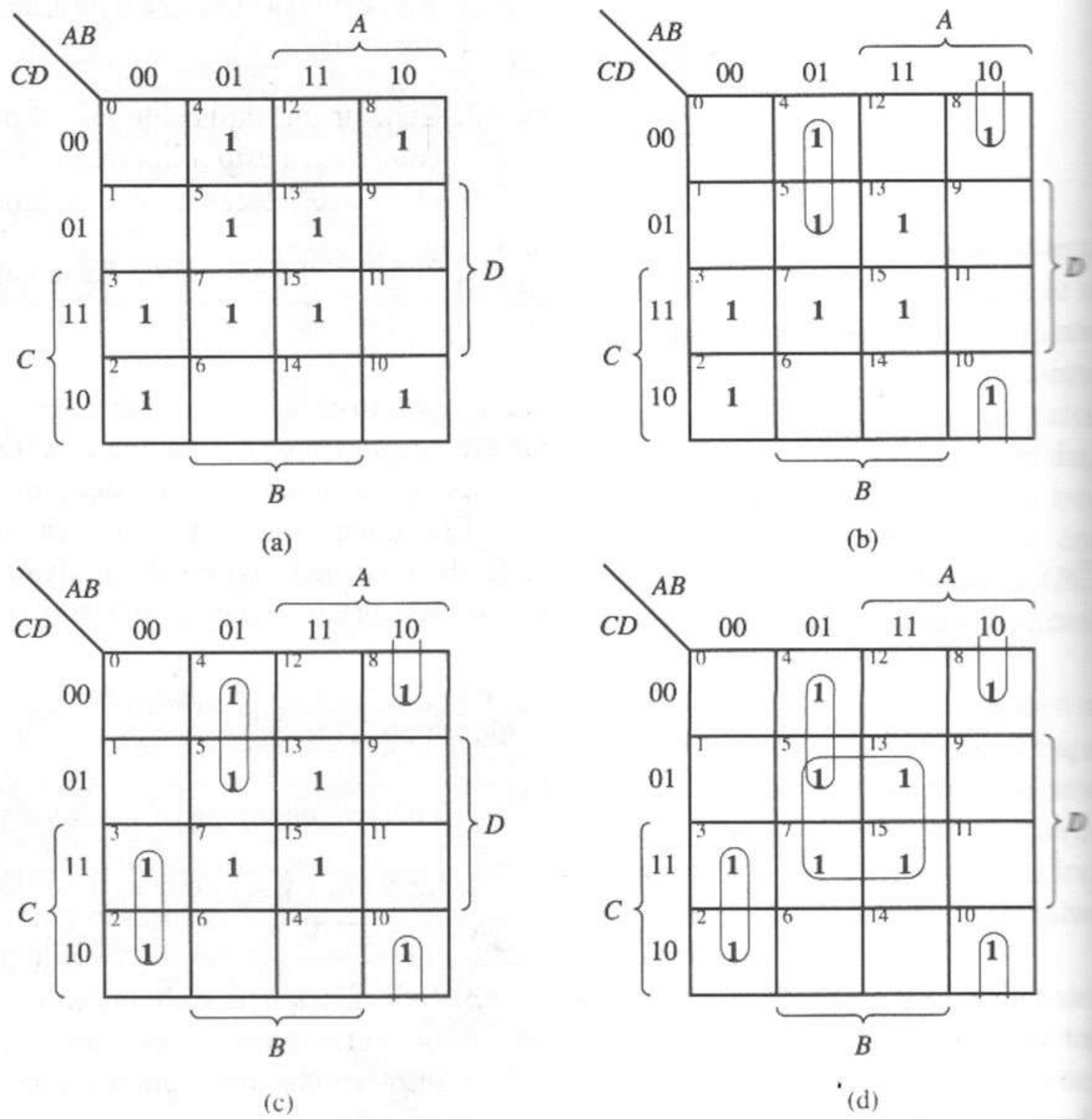


Figura 3.12 Mapas K que ilustran el algoritmo 3.1. **(a)** Graficación de la función. **(b)** Implicantes primos 4-5 y 8-10. **(c)** El implicante primo 2-3 cubre a m_2. **(d)** El implicante primo 5-7-13-15 completa la cubierta.

Ahora contamos las adyacencias para cada mintérmino. De la figura 3.12a vemos que cada uno de los mintérminos m_4 y m_8 tiene una adyacencia, cada uno de los mintérminos m_2, m_3, m_{10}, m_{13} y m_{15} tiene dos adyacencias y cada uno de los mintérminos m_5 y m_7 tiene tres adyacencias.

Como m_4 y m_8 tienen sólo una adyacencia, comenzamos con ellos. El implicante primo 4-5 es el único que cubre a m_4 y el implicante primo 8-10 es el único que cubre a m_8. Por tanto, agregamos estos dos implicantes primos a la cubierta encerrándolos en un círculo sobre el mapa K como en la figura 3.12b.

Al examinar los mintérminos restantes no cubiertos de la figura 3.12b vemos que cuatro de ellos tienen dos adyacencias, por lo que hacemos una selección arbitraria, digamos m_2. El mintérmino m_2 queda cubierto por los dos

implicantes primos 2-3 y 2-10. Como el implicante primo 2-3 cubre más de los mintérminos no cubiertos restantes, lo seleccionamos para la cubierta, encerrándolo en un círculo como se muestra en la figura 3.12c.

Como m_{13} y m_{15} tienen dos adyacencias cada uno, de nuevo hacemos una selección arbitraria, digamos m_{13}. El mintérmino m_{13} sólo queda cubierto por el implicante primo 5-7-13-15, de modo que agregamos este implicante primo a la cubierta encerrándolo en un círculo como se muestra en la figura 3.12d.

De la figura 3.12d vemos que ya están cubiertos todos los mintérminos. Por tanto, nuestra cubierta mínima es {4-5, 8-10, 2-3, 5-7-13-15}.

Obtenemos los términos producto que representan a los implicantes primos como sigue. En el mapa, el implicante primo 2-3 está fuera de A y B y dentro de C; por tanto, el término es $\bar{A}\bar{B}C$. El implicante primo 4-5 está fuera de A y C y dentro de B, por lo que el término es $A\bar{B}\bar{C}$. El implicante primo 5-7-13-15 está dentro de B y D, así que el término es BD. Por último, el implicante primo 8-10 se localiza dentro de A y fuera de B y de D; de modo que el término es $A\bar{B}\bar{D}$. Por tanto, la función minimizada es

$$f(A, B, C, D) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + BD + A\bar{B}\bar{D}$$

Ejemplo 3.11

Repetir el ejemplo 3.11 con el algoritmo 3.2.

Comenzamos encerrando en círculos todos los implicantes primos sobre el mapa K, como se muestra en la figura 3.13a. Los implicantes primos, enumerados según los mintérminos cubiertos, son {2-3, 3-7, 4-5, 5-7-13-15, 8-10, 2-10}.

En el paso 2 del algoritmo, seleccionamos los implicantes primos esenciales. En la figura 3.13a vemos que el implicante primo 4-5 es esencial porque cubre a m_4, el cual no queda cubierto por ningún otro implicante primo. Utilizamos un argumento similar para los implicantes primos 8-10 y 5-7-13-15. En consecuencia, estos implicantes primos esenciales deben ser parte de la cubierta para la función; los indicamos sobre el mapa K de la figura 3.13b.

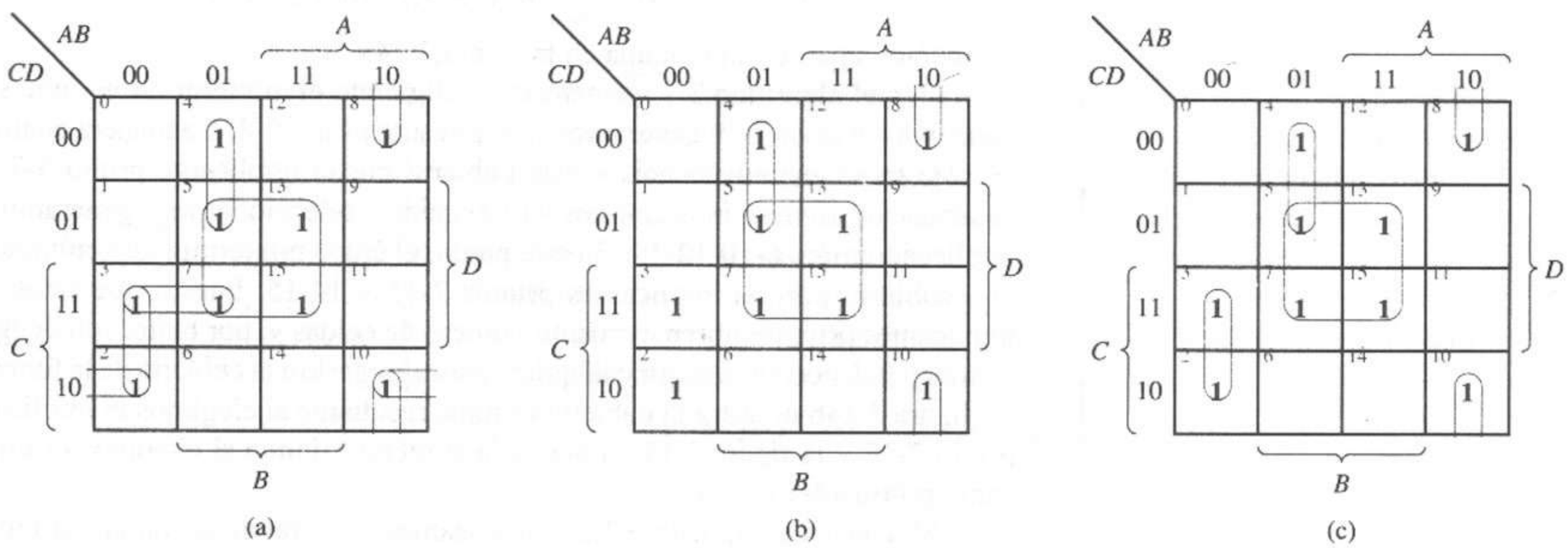


Figura 3.13 Mapas K que ilustran el algoritmo 3.2. **(a)** Todos los implicantes primos. **(b)** Implicantes primos esenciales. **(c)** Cubierta mínima.

En el paso 3 del algoritmo seleccionamos un número mínimo de implicantes primos que cubran los mintérminos restantes. Los únicos mintérminos de la figura 3.13b no cubiertos hasta el momento son m_2 y m_3. El mintérmino m_2 queda cubierto por los implicantes primos 2-3 y 2-10, mientras que m_3 queda cubierto por los implicantes primos 2-3 y 3-7. Si elegimos el implicante primo 2-3 cubrimos ambos mintérminos, agregando sólo un término producto a la cubierta. Al seleccionar 2-10 para cubrir a m_2, también necesitaría elegir otro implicante primo (3-7 o 2-3) para cubrir a m_3; esto agregaría dos términos producto adicionales a la cubierta. Por tanto, para obtener una realización mínima, elegimos el implicante primo 2-3, como indicamos en la figura 3.13c.

Así, nuestra cubierta mínima contiene al implicante primo 2-3 y a los implicantes primos esenciales 4-5, 5-7-13-15 y 8-10, que es el mismo resultado obtenido en el ejemplo anterior, indicado en la figura 3.12d.

Es muy importante que el lector comprenda que el uso de los mapas K sólo es un método conveniente para realizar reducciones en el álgebra de conmutación. Por ejemplo, si hubiéramos combinado los mintérminos 5-7-13-15 mediante el álgebra, en esencia habríamos desarrollado el siguiente proceso.

$$\begin{aligned} m_5 + m_7 + m_{13} + m_{15} &= \bar{A}B\bar{C}D + \bar{A}BCD + AB\bar{C}D + ABCD \\ &= (\bar{A}B\bar{C}D + \bar{A}BCD) + (AB\bar{C}D + ABCD) \\ &= \bar{A}BD + ABD \\ &= BD \end{aligned}$$

Así, varias veces combinamos los términos utilizando el teorema 6a, primero para eliminar la variable C y después para eliminar A. Así, al encerrar en un círculo cuatro cuadrados eliminamos dos variables. El lector debe verificar que obtenemos el mismo resultado de manera algebraica eliminando primero la variable A y después C.

EJEMPLO 3.12

Utilizar el mapa K para simplificar la siguiente función.

$$f(A, B, C, D) = \sum m(0, 5, 7, 8, 10, 12, 14, 15)$$

La función aparece en el mapa en la figura 3.14a.

Con el algoritmo 3.1, comenzamos eligiendo el mintérmino m_0, que sólo tiene una adyacencia, y generamos el implicante primo 0-8. De manera análoga, m_5 sólo tiene una adyacencia y está cubierto por el implicante primo 5-7. Al observar los mintérminos con dos adyacencias y seleccionar m_{10}, generamos el implicante primo 8-10-12-14. En este punto, el único mintérmino no cubierto es m_{15}, cubierto por los implicantes primos 7-15 y 14-15. Puesto que estos dos implicantes primos cubren el mismo número de celdas y, por tanto, son de igual complejidad, podemos elegir cualquiera para agregarlo a la cubierta de la función. La figura 3.14b muestra la cubierta mínima resultante si elegimos el implicante primo 7-15 y la figura 3.14c muestra la cubierta mínima si elegimos el implicante primo 14-15.

Si usamos el algoritmo 3.2, comenzamos encerrando en un círculo todos los implicantes primos, como en la figura 3.14d. De este mapa, podemos identificar como implicantes primos esenciales a 0-8, 5-7 y 8-10-12-14. Al seleccionar los implicantes primos esenciales para la cubierta, sólo queda sin cubrir

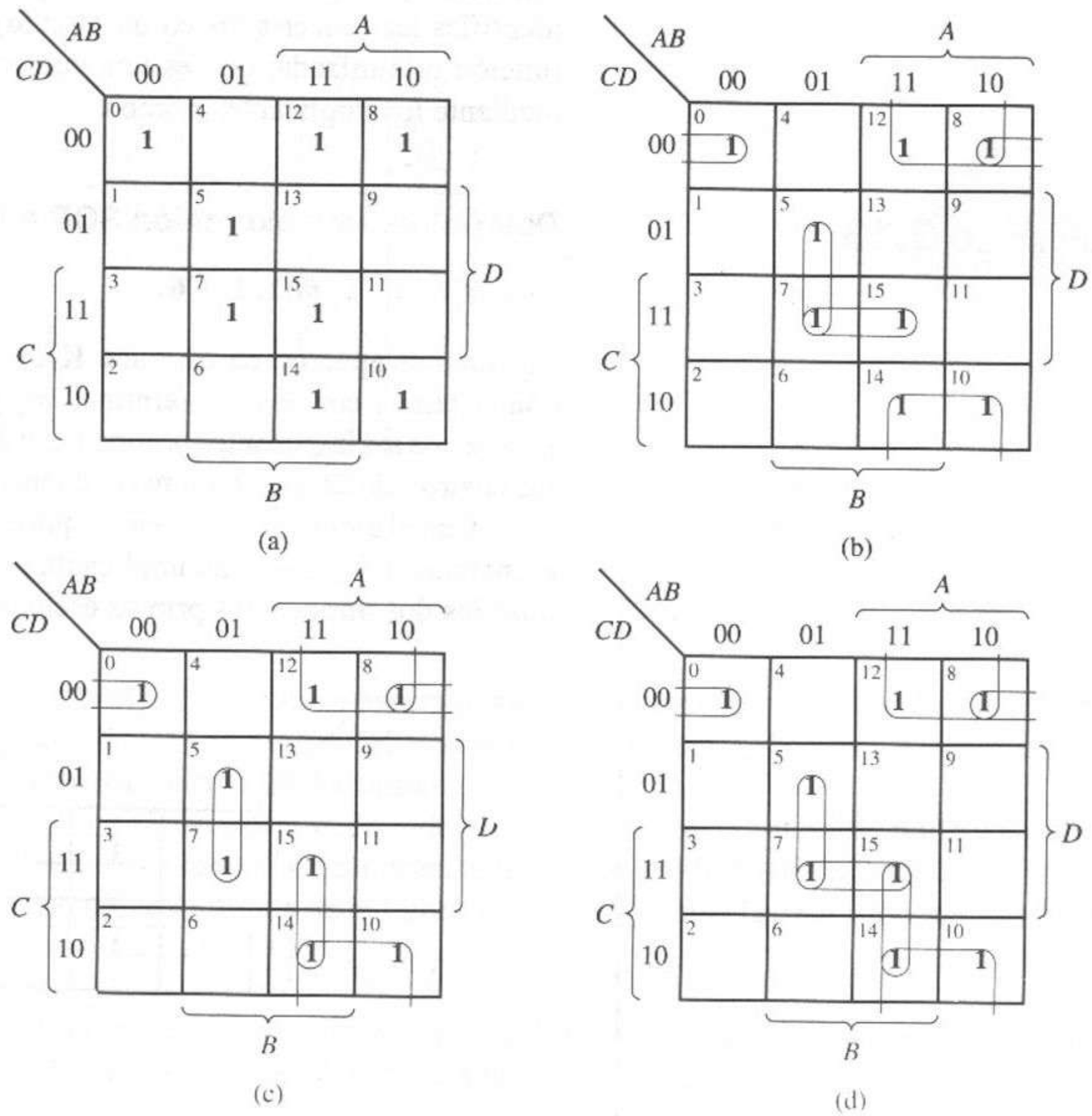


Figura 3.14 Mapas K para $f(A, B, C, D) = \Sigma m(0, 5, 7, 8, 10, 12, 14, 15)$. **(a)** Graficación de la función. **(b)** Cubierta mínima 1. **(c)** Cubierta mínima 2. **(d)** Todos los implicantes primos.

m_{15}. Por tanto, para completar la cubierta mínima, podemos seleccionar el implicante primo 7-15 o el 14-15 para cubrir a m_{15}, con lo que obtenemos las mismas dos soluciones de las figuras 3.14b y c obtenidas con el algoritmo 3.1.

Recuerde que podemos cubrir los mintérminos cualquier número de veces. En este ejemplo, m_8 queda cubierto dos veces en las dos soluciones. Es muy frecuente el uso múltiple de los mintérminos.

Ahora, de la cubierta mínima 1 de la figura 3.14b, podemos escribir la forma SOP mínima de la función $f(A, B, C, D)$ como

$$f(A, B, C, D) = \bar{B}\bar{C}\bar{D} + \bar{A}BD + A\bar{D} + BCD$$

y de la cubierta mínima 2 obtenemos

$$f(A, B, C, D) = \bar{B}\bar{C}\bar{D} + \bar{A}BD + A\bar{D} + ABC$$

Como ambas formas contienen el mismo número de términos y literales, cualquiera de ellas representa una expresión SOP mínima para la función.

Presentaremos rápidamente otros ejemplos. Cada uno tiene la función por minimizar, un mapa K con los implicantes primos de la cubierta mínima identificados (encerrados en un círculo), y, por último, una expresión para la función minimizada, que es una cubierta mínima de la función seleccionada mediante los implicantes primos.

EJEMPLO 3.13

Determinar una expresión SOP mínima para

$f(A,B,C) = \sum m(1,2,3,6)$.

La función aparece en el mapa K de la figura 3.15. Con el algoritmo 3.1, comenzamos con los mintérminos m_1 y m_6, con una adyacencia cada uno y generamos los implicantes primos 1-3 y 2-6, respectivamente. Como estos cubren los cuatro mintérminos, no necesitamos más implicantes primos.

Con el algoritmo 3.2, vemos que esta función tiene dos implicantes primos esenciales, 1-3 y 2-6, y un implicante primo adicional, 2-3, que no es necesario, pues los dos implicantes primos esenciales cubren la función.

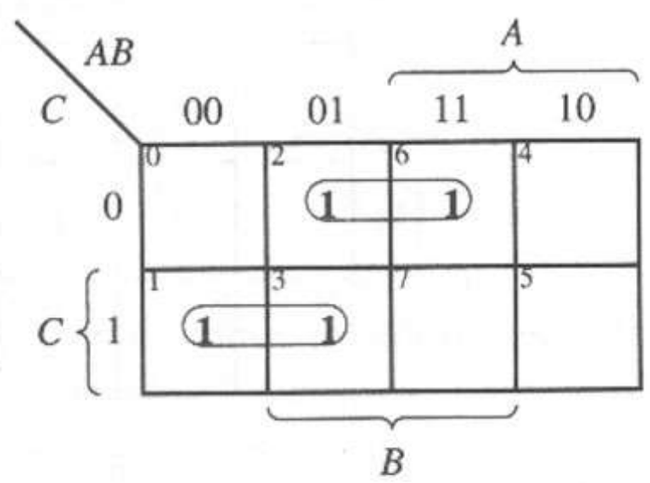


Figura 3.15 $f(A, B, C) = \sum m(1, 2, 3, 6) = \bar{A}C + B\bar{C}$.

Por tanto, la forma SOP mínima es la suma de los dos implicantes primos:

$$f(A, B, C) = \bar{A}C + B\bar{C}$$

EJEMPLO 3.14

Determinar una expresión SOP mínima para

$f(A,B,C,D) = \sum m(0,1,2,7,8,9,10,15)$.

Esta función, que aparece en la figura 3.16, tiene tres implicantes primos, 7-15, 0-1-8-9 y 0-2-8-10, todos los cuales son esenciales, por lo que forman la cubierta mínima de la función. Observe que el implicante primo 0-2-8-10 cubre las cuatro esquinas del mapa K. Las cuatro esquinas son adyacentes, ya que las filas inferior y superior son adyacentes (difieren en la variable C), al igual que las columnas extrema izquierda y extrema derecha (difieren en la variable A)

EJEMPLO 3.15

Determinar una expresión SOP mínima para

$f(A,B,C,D) = \sum m(0,4,5,7,8,10,14,15)$.

Del mapa K de la figura 3.17a, vemos que cada mintérmino está cubierto por dos implicantes primos, por lo que ninguno de los dos es esencial. Además,

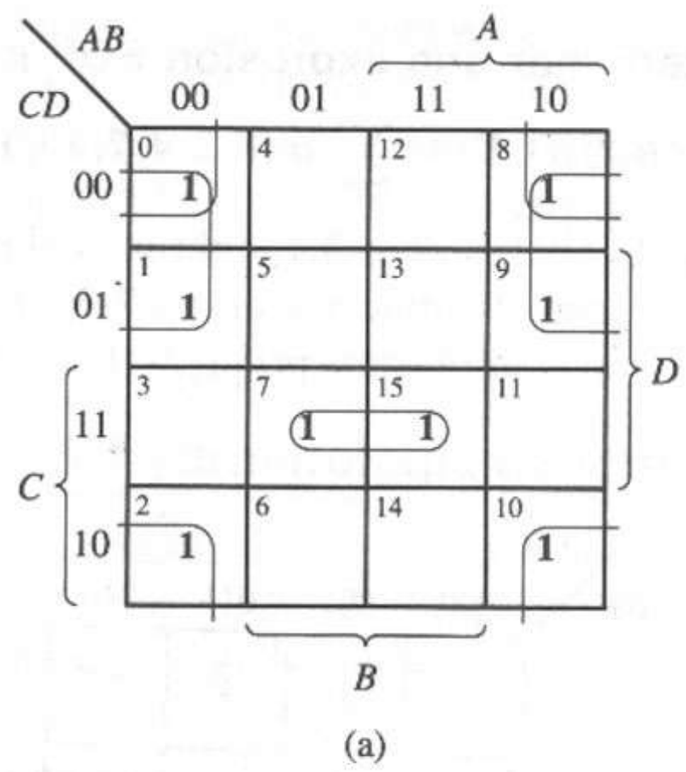


Figura 3.16 $f(A, B, C, D) = \bar{B}D + \bar{B}\bar{C} + BCD$.

cada mintérmino tiene exactamente dos adyacencias. Esta condición se conoce como *ciclo* en el mapa K. Cuando hay un ciclo, debemos romperlo haciendo una selección inicial arbitraria.

Comenzamos cubriendo el mintermino m_0. Vemos que podemos cubrirlo con cualquiera de los implicantes primos 0-4 o 0-8. Al seleccionar 0-4, obtenemos la cubierta mínima de la figura 3.17b, para la cual la expresión mínima SOP es

$$f(A, B, C, D) = \bar{A}\bar{C}\bar{D} + \bar{A}BD + ABC + A\bar{B}\bar{D}$$

La selección del implicante primo 0-8 para cubrir a m_0 nos lleva a la cubierta mínima de la figura 3.17c, para la cual la expresión mínima SOP es

$$f(A, B, C, D) = \bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C} + BCD + AC\bar{D}$$

Así, obtenemos dos expresiones mínimas SOP totalmente diferentes para la función, con el mismo número de términos y literales y, por tanto, con el mismo costo.

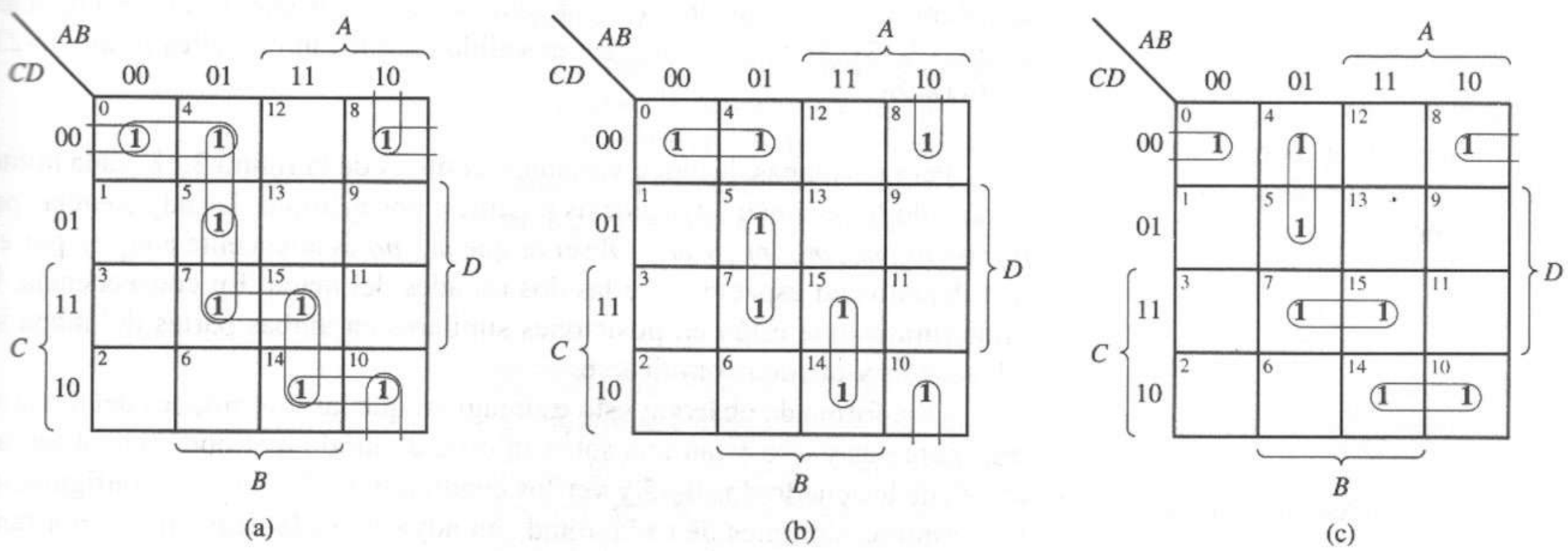


Figura 3.17 Función sin implicantes primos esenciales. **(a)** Todos los implicantes primos. **(b)** Cubierta mínima 1. **(c)** Cubierta mínima 2.

EJEMPLO 3.16

Determinar una expresión SOP mínima para

$$f(A,B,C,D,E) = \sum m(0,2,4,7,10,12,13,18,23,26,28,29).$$

Usando el algoritmo 3.1, cubrimos el mintérmino m_0 con el implicante primo 0-4, m_7 con el implicante primo 7-23, m_{13} con el implicante primo 12-13-28-29 y m_{10} con el implicante primo 2-10-18-26. Al indicar esto sobre el mapa K de la

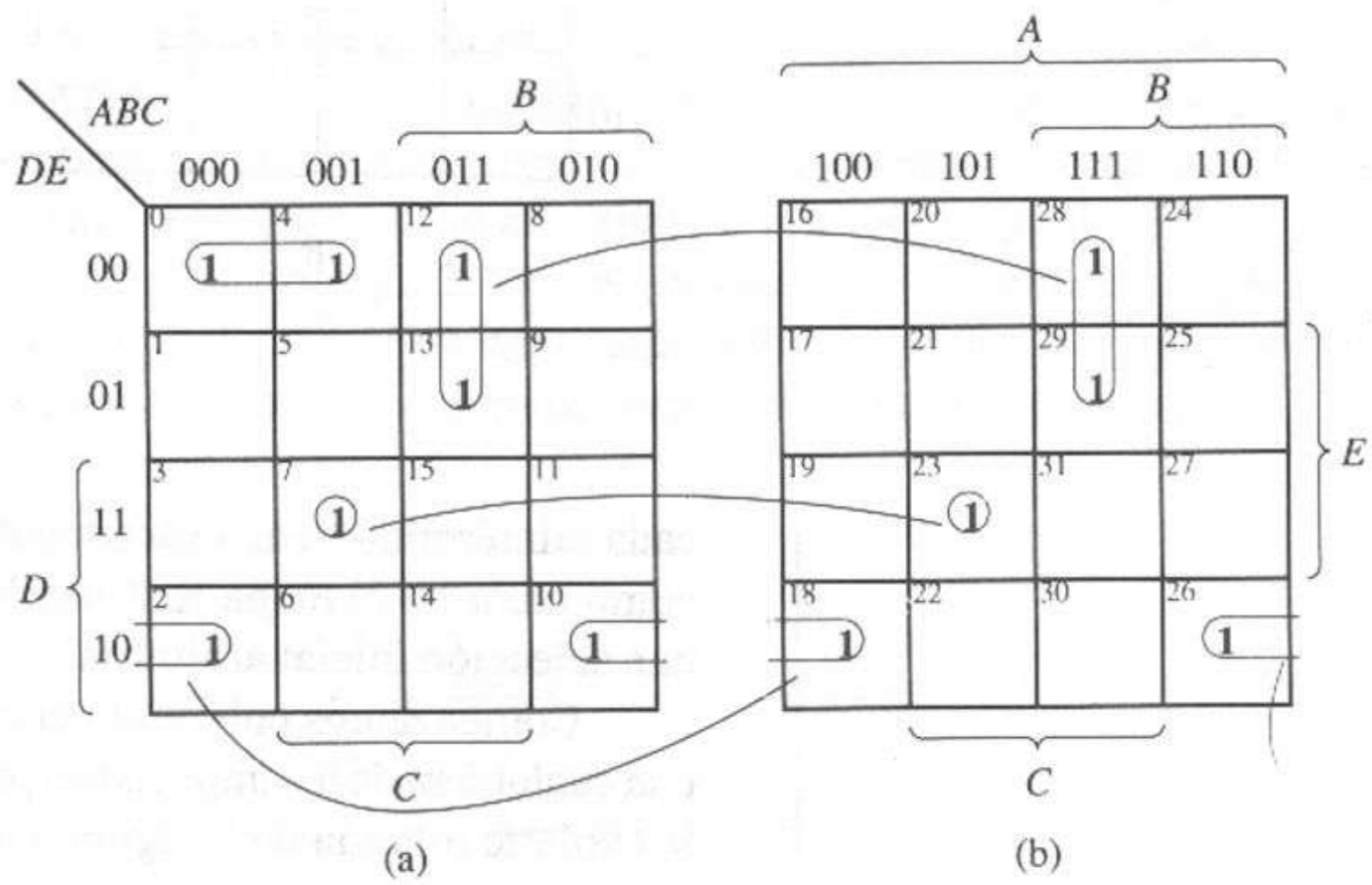


Figura 3.18 Minimización de una función de cinco variables.

figura 3.18, vemos que los cuatro implicantes primos son esenciales y forman una cubierta completa de la función. La forma SOP mínima de la función es

$$f(A, B, C, D, E) = \bar{A}\bar{B}\bar{D}\bar{E} + BC\bar{D} + \bar{B}CDE + \bar{C}D\bar{E}$$

Observe que hay que tener cuidado al identificar las adyacencias entre las dos mitades de un mapa de cinco variables. Por ejemplo, el implicante 12-13 es adyacente al implicante 28-29 y, por tanto, se combinan para formar el implicante primo 12-13-28-29. Esto también es válido para los implicantes primos 7-23 y 2-10-18-26.

Para los mapas de cinco variables, como el de la figura 3.18, cada mintérmino sólo tiene cinco adyacencias posibles; por ejemplo, las adyacencias para m_8 son m_0, m_9, m_{10}, m_{12} y m_{24}. Observe que m_{16} *no* es adyacente a m_8; es por ello que dejamos un espacio entre las dos mitades del mapa. En consecuencia, los mintérminos que están en posiciones similares en ambas partes del mapa son adyacentes y pueden combinarse.

Otra forma de observar esto es imaginar que las dos mitades del mapa son transparentes y que están una sobre la otra, de modo que podemos observar a través de los cuadrados 0-15 y ver los cuadrados 16-31. En esta configuración, los términos similares de cada mitad son adyacentes lógicamente y, por tanto, se pueden combinar. Por ejemplo, la combinación de mintérminos m_{13} y m_{29} se puede agrupar para eliminar la variable A, al igual que los mintérminos m_7 y m_{23}, m_2 y m_{18}, etcétera.

3.6 Forma POS mediante mapas K

Hasta ahora nos hemos centrado en el uso del mapa K para minimizar una función en forma SOP. Podemos seguir un procedimiento idéntico para minimizar una función en forma POS (producto de sumas). Además, podemos aplicar todas las técnicas que hemos aprendido para combinar mintérminos al caso de los maxtérminos y así obtener una forma POS mínima.

3.6.1 Terminología general para las formas POS

De una manera similar a la minimización de las formas SOP, definimos los términos *implicado, implicado primo, implicado primo esencial* y *cubierta* para utilizarlos en la deducción de formas POS mínimas.

Un *implicado* es un término suma, es decir, una suma de una o más literales, que puede servir para cubrir maxtérminos de la función. Observe que un implicado representa una combinación de entradas donde la función asume el valor 0. En el mapa K, un implicado es un grupo de maxtérminos adyacentes, o cuadrados 0. Un *implicado primo* es un implicado no cubierto por ningún otro implicado de la función. En el mapa K, un implicado primo es un grupo de maxtérminos adyacentes que no está cubierto por un grupo mayor de maxtérminos. Un *implicado primo esencial* es un implicado primo que cubre al menos un maxtérmino no cubierto por algún otro implicado primo. En el mapa K, un implicado primo esencial cubre al menos un maxtérmino encerrado en un círculo una sola vez. Una *cubierta* de una función es un conjunto de implicados tales que cada maxtérmino de la función está contenido en (queda cubierto por) al menos un implicado primo.

3.6.2 Algoritmos para la deducción de formas POS mínimas a partir de mapas K

Podemos modificar fácilmente los algoritmos 3.1 y 3.2 para la generación de cubiertas mínimas de una función, de las cuales se obtienen expresiones SOP mínimas, para obtener cubiertas mínimas de las cuales deducir las formas POS. El proceso de agrupación de celdas en grupos maximales y la selección posterior de grupos de celdas para cubrir la función es idéntico, excepto que ahora las celdas representan maxtérminos en vez de mintérminos, y los grupos son implicados en vez de implicantes. Por tanto, tenemos los siguientes dos algoritmos para generar una cubierta mínima de los maxtérminos de una función.

Algoritmo 3.3

1. Contar el número de adyacencias para cada maxtérmino del mapa K.
2. Seleccionar un maxtérmino no cubierto con el menor número de adyacencias. Se elige en forma arbitraria si existen varias opciones.

3. Generar un implicado primo para este maxtérmino y colocarlo en la cubierta. Si el maxtérmino está cubierto por más de un implicado primo, seleccionar el que cubra más maxtérminos no cubiertos.

4. Repetir los pasos 2 y 3 hasta cubrir todos los maxtérminos.

Algoritmo 3.4

1. Encerrar en un círculo todos los implicados primos del mapa K.

2. Identificar y seleccionar todos los implicados primos esenciales para la cubierta.

3. Seleccionar un subconjunto mínimo de los implicados primos restantes para completar la cubierta; es decir, cubrir los maxtérminos no cubiertos por los implicados primos esenciales.

Después de obtener una cubierta mínima de la función, podemos deducir la expresión POS mínima escribiendo cada implicado primo como un término suma y formando después el producto de los términos suma.

Demostraremos el proceso de minimización con un ejemplo detallado y después veremos varios ejemplos más rápidamente.

EJEMPLO 3.17

Determinar la forma POS mínima para la función

$$f(A,B,C,D) = \prod M(0,1,2,3,6,9,14).$$

Esta función aparece en el mapa de la figura 3.19a.

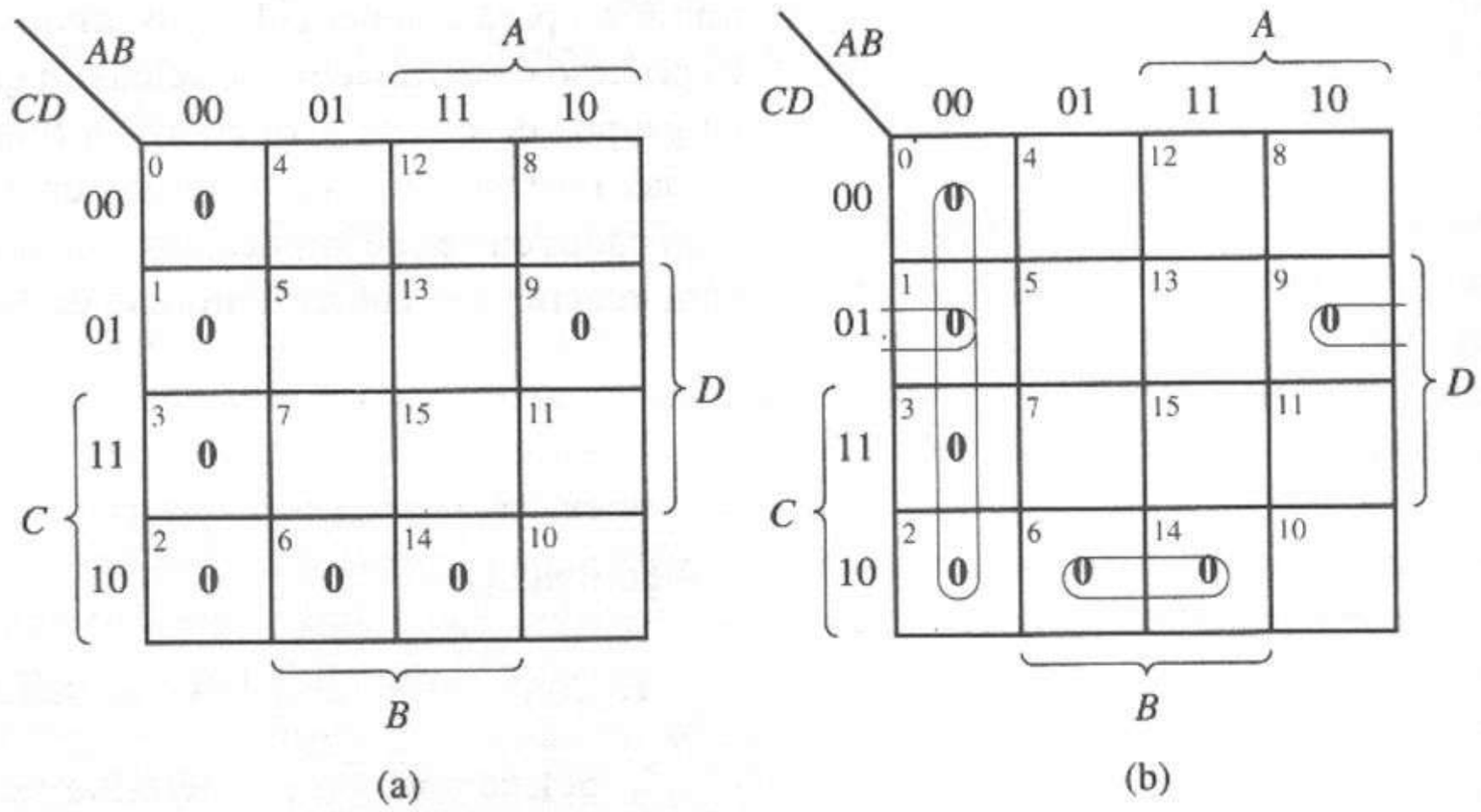


Figura 3.19 Mapas K del ejemplo 3.17. (a) Graficación de la función. (b) Cubierta mínima.

Siguiendo el algoritmo 3.3, comenzamos por contar las adyacencias y observamos que cada uno de los maxtérminos M_9 y M_{14} tiene una adyacencia. Por tanto, cubrimos M_9 con el implicado primo 1-9, y cubrimos M_{14} con el implicado primo 6-14. Después nos fijamos en los maxtérminos con dos adyacencias, seleccionamos el maxtérmino M_0 y lo cubrimos con el implicado primo 0-1-2-3, lo que completa la cubierta. Esta cubierta mínima aparece en la figura 3.19b.

Si utilizamos el algoritmo 3.4 para obtener la cubierta mínima, llegamos al mismo resultado. Al examinar la figura 3.19b, podemos ver que los tres implicados primos 0-1-2-3, 1-9 y 6-14 son esenciales y que cubren todos los maxtérminos de la función. Existe un implicado primo adicional, 2-6, que no es necesario para la cubierta.

Obtenemos la forma POS mínima a partir de la cubierta mínima como sigue. El implicado primo 0-1-2-3 representa el área del mapa donde A y B son 0. Por tanto, el término suma para este implicado primo esencial es $(A + B)$. En otras palabras, $f(A, B, C, D) = 0$ cuando $(A + B) = 0$, lo que sucede cuando $A = 0$ y $B = 0$. El implicado primo 1-9 representa la porción del mapa donde B es 0, C es 0 y D es 1. Por tanto, el término suma para este implicado primo es $(B + C + \overline{D})$. El último implicado primo es 6-14, que representa el área del mapa K donde B es 1, C es 1 y D es 0. Por tanto, el término suma para este implicado primo es $(\overline{B} + \overline{C} + D)$. Así, la forma POS mínima para esta función es

$$f(A, B, C, D) = (A + B)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$$

A algunos diseñadores digitales les desagrada trabajar con maxtérminos e implicados primos y prefieren simplificar sobre el mapa K utilizando uno de los algoritmos para la obtención de formas SOP mínimas. Podemos aprovechar los métodos SOP para obtener una forma POS mínima de una función, f, trabajando con $\overline{f}$ en vez de f. Comenzamos graficando el complemento de la función $\overline{f}$ en el mapa K. Esto convierte los maxtérminos (ceros) en mintérminos (unos). Después utilizamos uno de los procedimientos SOP para deducir una expresión SOP mínima para $\overline{f}$. Después, complementamos esta expresión SOP y aplicamos el teorema de DeMorgan (teorema 8) para obtener la forma POS deseada de f. Así, minimizamos $\overline{f}$ como una función SOP y después la complementamos para obtener f. Podemos resumir este procedimiento como sigue:

Algoritmo 3.5

1. Graficar el complemento de la función $\overline{f}$ en el mapa K.
2. Utilizar el algoritmo 3.1 o 3.2 para obtener una expresión SOP mínima para $\overline{f}$.
3. Complementar la expresión y aplicar el teorema de DeMorgan para obtener una expresión POS mínima.

EJEMPLO 3.18

Repetir el ejemplo 3.17 con el algoritmo 3.5.

Dada la función

$$f(A, B, C, D) = \prod M(0, 1, 2, 3, 6, 9, 14)$$

que se muestra en la figura 3.19a, comenzamos graficando su complemento en otro mapa K:

$$\bar{f}(A, B, C, D) = \sum m(0, 1, 2, 3, 6, 9, 14)$$

Esto aparece en el mapa K de la figura 3.20a. Observe que este mapa es idéntico al de la figura 3.19a, pero con ceros en vez de unos.

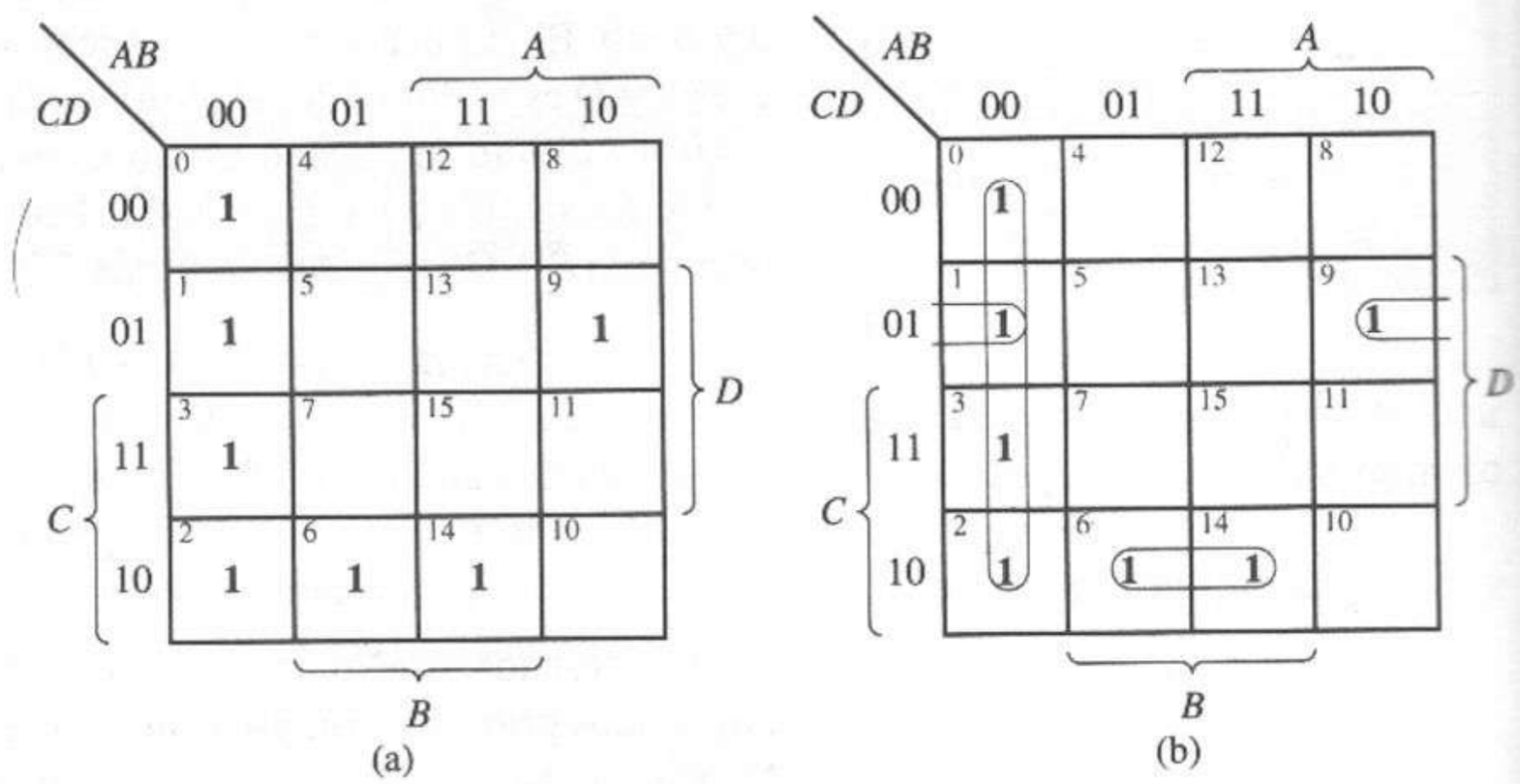


Figura 3.20 Mapa K de $\bar{f}(A, B, C, D)$. **(a)** Mintérminos de la función. **(b)** Cubierta mínima.

Con el algoritmo 3.1 o el 3.2, obtenemos la cubierta mínima {1-9, 6-14, 0-1-2-3}. Observe que el orden de los términos y la cubierta resultante son los mismos que en el ejemplo 3.17, excepto que aquí la cubierta representa los implicantes primos de $\bar{f}$, en vez de los implicados primos de f.

Ahora escribimos la expresión SOP mínima para $\bar{f}$ a partir de los implicados primos de la cubierta mínima:

$$\bar{f}(A, B, C, D) = \bar{A}\bar{B} + \bar{B}\bar{C}D + BC\bar{D}$$

Por último, complementamos la expresión y aplicamos el teorema de DeMorgan para obtener la forma POS mínima de $f(A, B, C, D)$:

$$\begin{aligned} f(A, B, C, D) &= \overline{\bar{A}\bar{B} + \bar{B}\bar{C}D + BC\bar{D}} \\ &= (\overline{\bar{A}\bar{B}})(\overline{\bar{B}\bar{C}D})(\overline{BC\bar{D}}) \\ &= (A + B)(B + C + \bar{D})(\bar{B} + \bar{C} + D) \end{aligned}$$

La principal ventaja de este método es que podemos usar las reglas SOP para la simplificación en el mapa K de las funciones tanto SOP como POS. Sin embargo, necesitamos complementar la expresión y aplicar el teorema de

DeMorgan para obtener una forma mínima POS. Los siguientes ejemplos utilizan ambos métodos.

EJEMPLO 3.19

Determinar una expresión POS mínima para

Los maxtérminos de la función están indicados en el mapa K de la figura 3.21a. Con el algoritmo 3.3, cubrimos primero el maxtérmino M_3 con el implicado primo 3-11, M_4 con 4-6-12-14 y después M_8 con 8-9. La cubierta mínima resultante aparece en la figura 3.21a. Obtenemos la forma POS mínima escribiendo el producto de términos suma correspondiente a los implicados primos, donde 3-11 representa a $(B + \overline{C} + \overline{D})$, 4-6-12-14 representa a $(\overline{B} + D)$ y 8-9 representa a $(\overline{A} + B + C)$. La expresión POS resultante es

$$f(A, B, C, D) = (\bar{B} + D)(B + \bar{C} + \bar{D})(\bar{A} + B + C)$$

Para utilizar el algoritmo 3.5, graficamos el complemento de la función, como en la figura 3.21b, y después formamos la misma combinación de celdas: 3-11, 4-6-12-14 y 8-9. Escribimos la suma de estos implicantes primos para obtener

$$\bar{f}(A, B, C, D) = B\bar{D} + \bar{B}CD + A\bar{B}\bar{C}$$

Complementamos la expresión y aplicamos el teorema de DeMorgan,

$$\begin{aligned} f(A, B, C, D) &= \overline{B\bar{D} + \bar{B}CD + A\bar{B}\bar{C}} \\ &= (\overline{B\bar{D}})(\overline{\bar{B}CD})(\overline{A\bar{B}\bar{C}}) \\ &= (\bar{B} + D)(B + \bar{C} + \bar{D})(\bar{A} + B + C) \end{aligned}$$

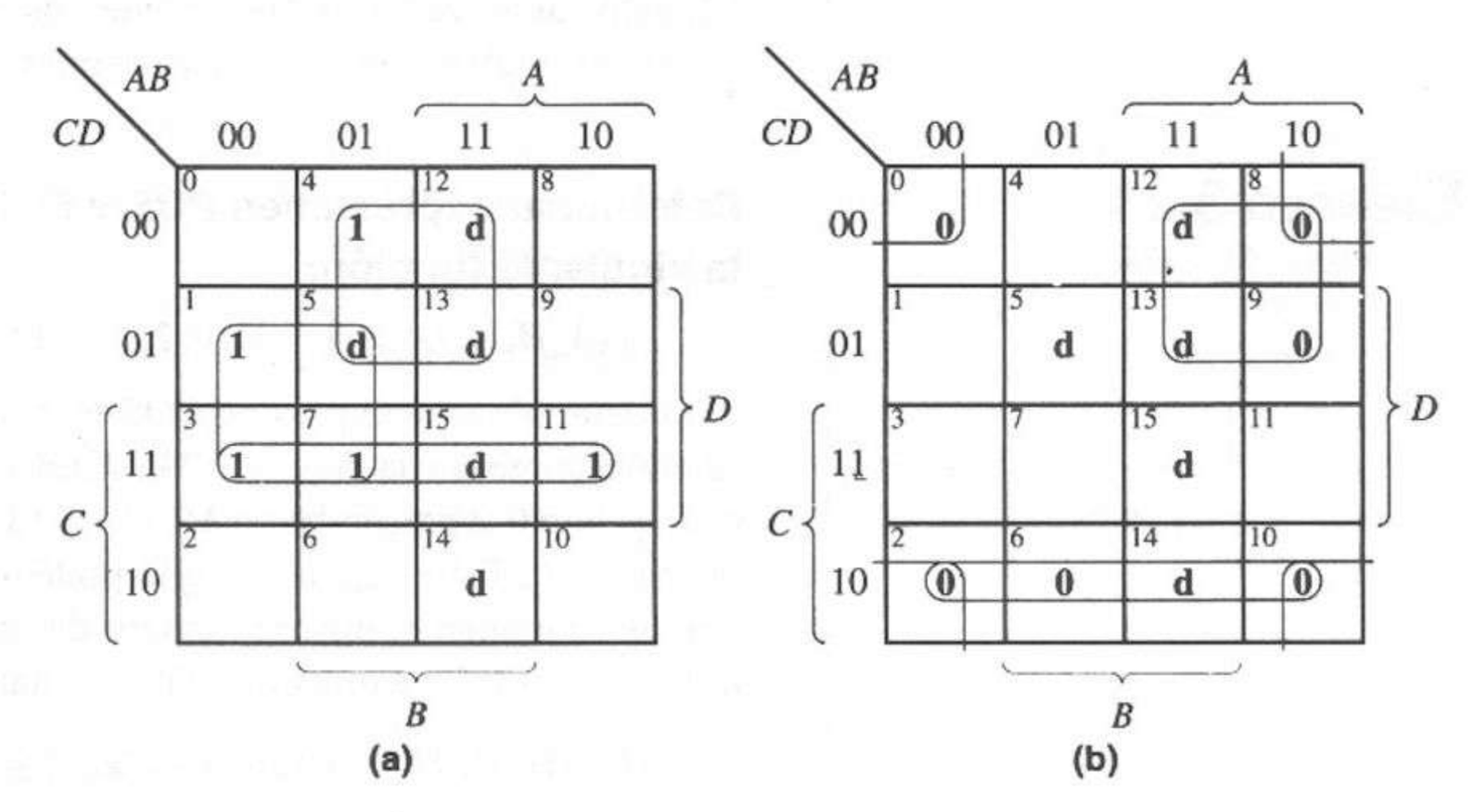


Figura 3.21 Cubiertas mínimas de $f(A, B, C, D) = \Pi M(3, 4, 6, 8, 9, 11, 12, 14)$ y su complemento. **(a)** $f(A, B, C, D)$. **(b)** $\bar{f}(A, B, C, D)$.

EJEMPLO 3.20

Deducir una expresión POS mínima para la función

La figura 3.22 muestra la función y su cubierta mínima sobre el mapa K. La cubierta mínima contiene cinco implicados primos, todos ellos esenciales.

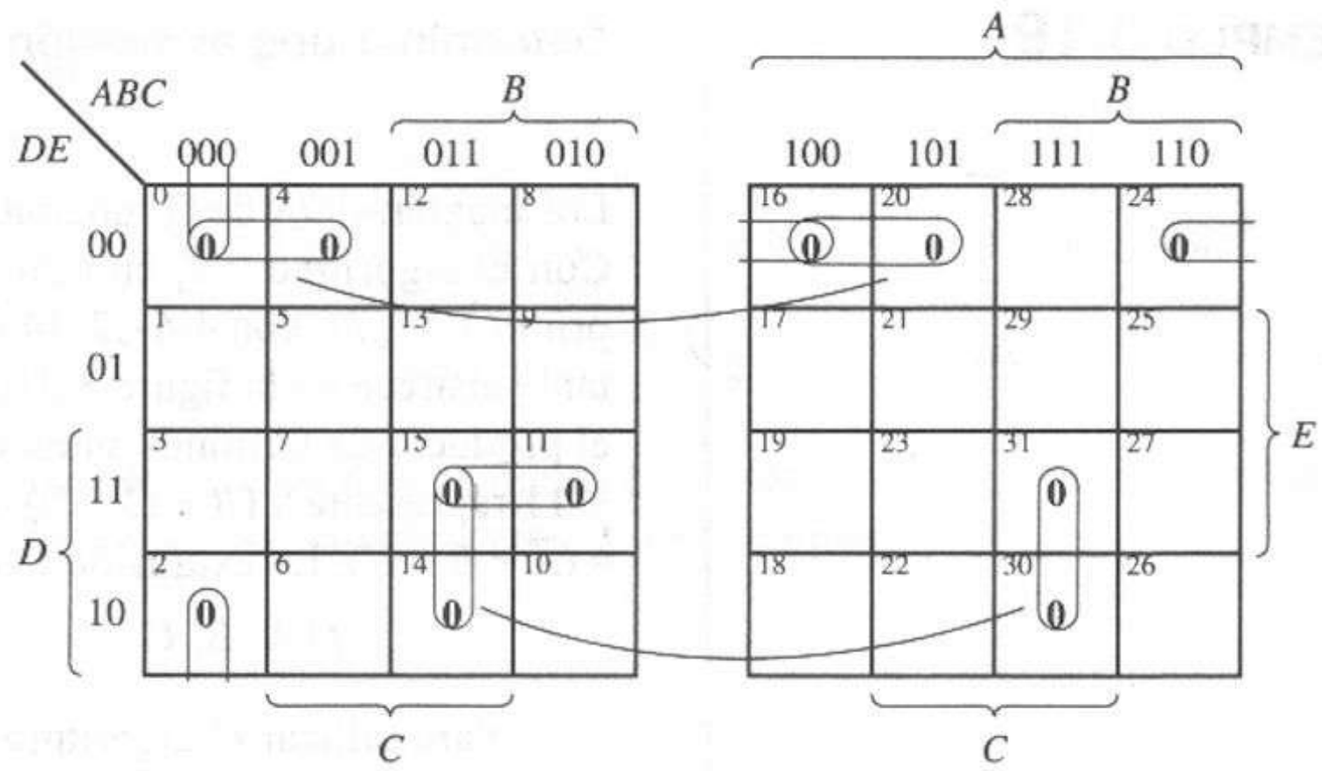


Figura 3.22 Determinación de la expresión POS mínima para una función de 5 variables.

Al escribir estos implicados primos como sumas y considerar su producto, obtenemos

$$f(A, B, C, D, E) = (A + B + C + E)(B + D + E)(\bar{B} + \bar{C} + \bar{D})$$
$$(A + \bar{B} + \bar{D} + \bar{E})(\bar{A} + C + D + E)$$

El lector debe verificar que obtenemos el mismo resultado al determinar una expresión SOP mínima para $\bar{f}$ y después complementarla para obtener f.

EJEMPLO 3.21

Determinar expresiones POS y SOP mínimas para la siguiente función:

$$f(A,B,C,D) = \prod M(0,2,3,9,11,12,13,15)$$

Para determinar la expresión POS mínima, graficamos los maxtérminos de la función como en la figura 3.23a. Con el algoritmo 3.3, generamos el implicado primo 0-2 para cubrir a M_2, el 12-13 para cubrir a M_{12} y el 9-11-13-15 para cubrir a M_9. Esto deja a M_3, que podemos cubrir con 3-11 o con 2-3. Como ambos contienen el mismo número de literales, seleccionamos uno de manera arbitraria, 2-3. La expresión POS resultante es entonces

$$f(A, B, C, D) = (A + B + D)(\bar{A} + \bar{B} + C)(\bar{A} + \bar{D})(A + B + \bar{C})$$

También podemos generar la forma POS mínima graficando $\bar{f}$ como en la figura 3.23b, y deduciendo después una expresión SOP mínima para $\bar{f}$:

$$\bar{f}(A, B, C, D) = \bar{A}\bar{B}\bar{D} + AB\bar{C} + AD + \bar{A}\bar{B}C$$

Al complementar, obtenemos

$$\begin{aligned} f(A, B, C, D) &= \overline{\bar{A}\bar{B}\bar{D} + AB\bar{C} + AD + \bar{A}\bar{B}C} \\ &= (\overline{\bar{A}\bar{B}\bar{D}})(\overline{AB\bar{C}})(\overline{AD})(\overline{\bar{A}\bar{B}C}) \\ &= (A + B + D)(\bar{A} + \bar{B} + C)(\bar{A} + \bar{D})(A + B + \bar{C}) \end{aligned}$$

A fin de generar una expresión SOP mínima para $f(A, B, C, D)$, graficamos los mintérminos de la función en vez de los maxtérminos, como en la figura 3.23c. Los algoritmos 3.1 o 3.2 producen la cubierta mínima de la figura 3.23c. Al considerar la suma de los implicados primos obtenemos

$$f(A, B, C, D) = \bar{A}\bar{C}D + A\bar{B}\bar{D} + \bar{A}B + BC\bar{D}$$

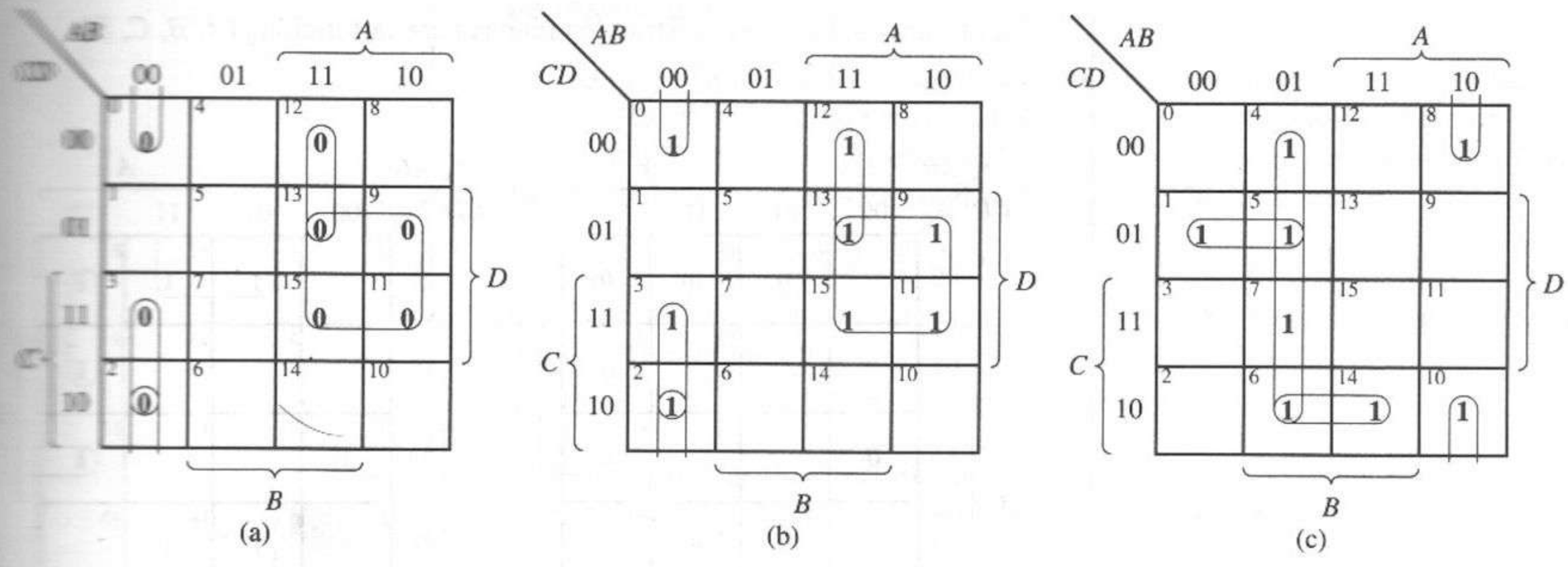


Figura 3.23 Deducción de las formas POS y SOP de una función. **(a)** Maxtérminos de f. **(b)** Mintérminos de $\bar{f}$. **(c)** Mintérminos de f.

El ejemplo 3.21 ilustra un punto importante. Al partir de cualquier descripción de una función, podemos generar una expresión SOP o una expresión POS mínima. Para generar una expresión SOP mínima trabajamos con los mintérminos de la función, y para obtener una expresión POS mínima trabajamos con los maxtérminos. Después de graficar los maxtérminos de una función en el mapa K, como en el ejemplo 3.21, obtenemos en forma automática un mapa de mintérminos, y viceversa. Por tanto, el formato de la función inicial no necesariamente afecta la elección de una forma SOP o POS para la función minimizada.

3.7 Funciones con especificación incompleta

Si hay términos prescindibles, agregamos una regla a las analizadas para minimizar funciones mediante mapas. Recordemos que los prescindibles pueden ser, por definición, 0 o 1. Por tanto, al minimizar los términos en forma SOP o POS, elegimos los prescindibles como 0 o 1 si, al hacerlo, el conjunto de cuadrados del mapa que se pueden agrupar es mayor de lo que sería sin los prescindibles. Después, al deducir la cubierta mínima, ignoramos los prescindibles y sólo elegimos los implicantes o implicados primos suficientes para cubrir los términos dados. En otras palabras, en lo que se refiere a los pres-

cindibles, podemos considerarlos (o no) según que nos ayuden (o no) a simplificar la función.

EJEMPLO 3.22

Minimizar la siguiente función en sus formas SOP y POS mediante los mapas K.

$$f(A,B,C,D) = \sum m(1,3,4,7,11) + d(5,12,13,14,15)$$
$$= \prod M(0,2,6,8,9,10) \cdot D(5,12,13,14,15)$$

Las figuras 3.24a y b muestran los mapas para la función $f(A, B, C, D)$.

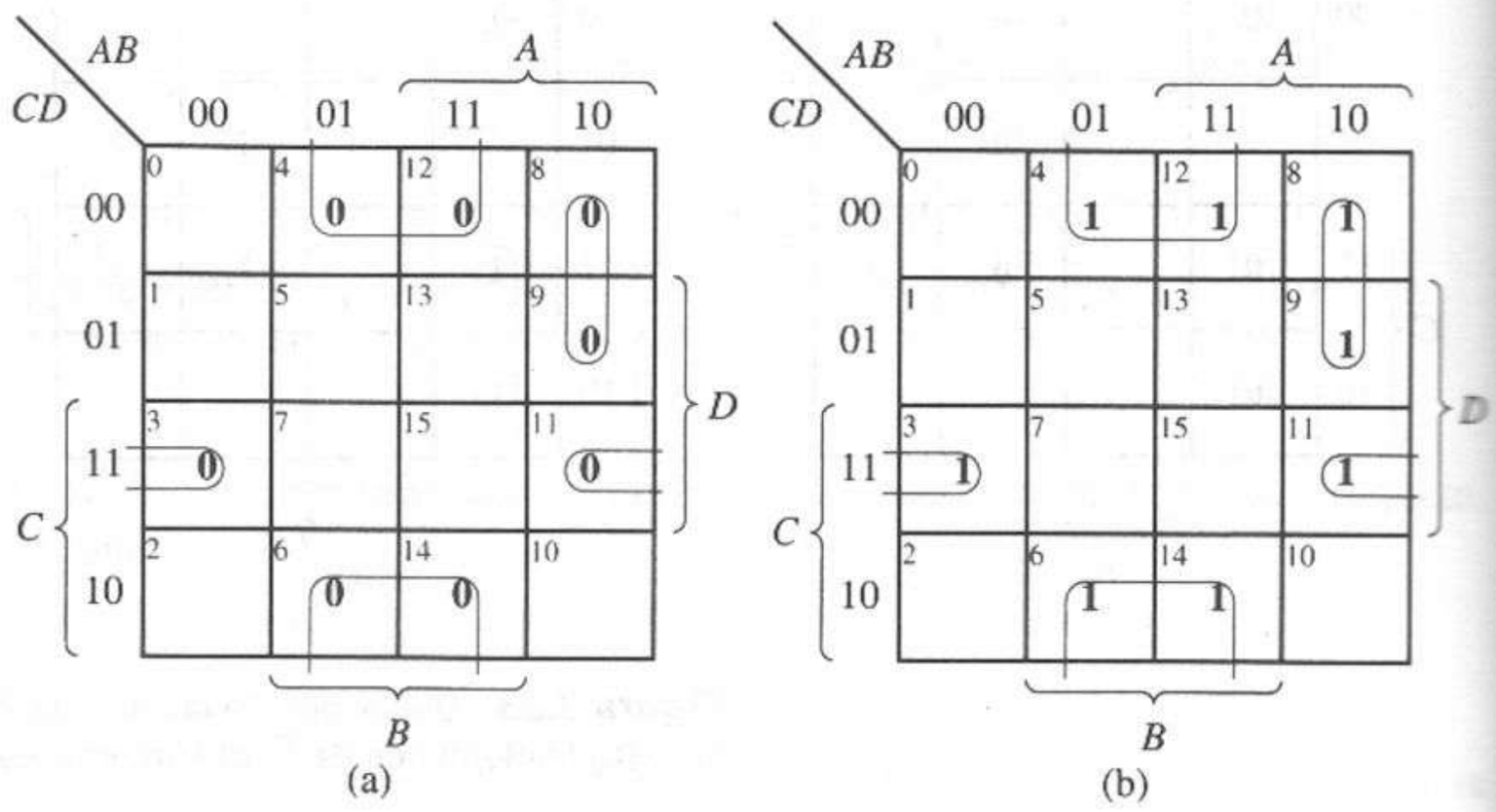


Figura 3.24 Mapas K del ejemplo 3.22. **(a)** Suma de productos. **(b)** Producto de sumas.

La forma SOP mínima obtenida mediante el mapa de la figura 3.24a es

$$f(A, B, C, D) = B\bar{C} + \bar{A}D + CD$$

La forma POS mínima obtenida mediante el mapa de la figura 3.24b es

$$f(A, B, C, D) = (B + D)(\bar{C} + D)(\bar{A} + C)$$

Observe que, si aplicamos el álgebra booleana a la forma POS mínima para obtener una forma SOP, tenemos

$$f(A, B, C, D) = \bar{A}B\bar{C} + \bar{A}D + CD$$

que no es idéntica a la forma SOP mínima obtenida del mapa. Esta situación es frecuente cuando hay prescindibles, pues éstos se pueden utilizar de manera diferente al optimizar cada expresión. En este caso, usamos varios términos prescindibles (5, 12, 13) como unos para deducir la forma SOP mínima y como ceros para deducir la forma POS mínima. Sin embargo, en cada caso, los mintérminos y maxtérminos *requeridos* tienen una implantación correcta.

El siguiente ejemplo ilustra cómo aparecen los prescindibles y su uso.

EJEMPLO 3.23

Diseñar un circuito lógico con una entrada BCD de 4 bits y una sola salida que servirá para distinguir los dígitos que son mayores o iguales que 5 de los menores que 5. La entrada será la representación BCD de los dígitos decimales 0, 1,..., 9, y la salida será 1 si la entrada es 5, 6, 7, 8 o 9 y 0 si la entrada es menor que 5.

La figura 3.25a muestra el diagrama de bloques del circuito, mientras que la 3.25b es la tabla de verdad para esta operación. Observe que los prescindibles aparecen en la tabla porque estas entradas en particular no representan dígitos BCD y, por tanto, no pueden ocurrir. Así, la función de salida f es

$$f(A, B, C, D) = \sum m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$

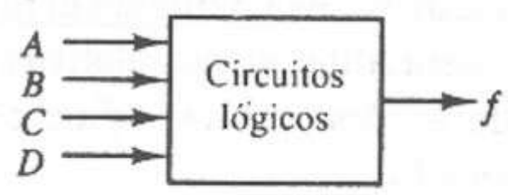
Esta función se indica sobre el mapa de la figura 3.26a. Del mapa, obtenemos la forma SOP mínima:

$$f(A, B, C, D) = A + BD + BC$$

El lector debe verificar que al combinar los maxtérminos y los prescindibles, como en la figura 3.26b, podemos obtener la forma POS mínima:

$$f(A, B, C, D) = (A + B)(A + C + D).$$

Observe que esta función es más sencilla que si no hubiéramos incluido los prescindibles. Además, observe en la figura 3.26a que utilizamos todos los prescindibles, es decir, todos fueron unos, mientras que en la figura 3.26b no utilizamos los prescindibles, es decir, todos fueron ceros. Esto no siempre ocurre.



(a)

ABCD	Mintérmino	$f(A, B, C, D)$
0000	0	0
0001	1	0
0010	2	0
0011	3	0
0100	4	0
0101	5	1
0110	6	1
0111	7	1
1000	8	1
1001	9	1
1010	10	*d*
1011	11	*d*
1100	12	*d*
1101	13	*d*
1110	14	*d*
1111	15	*d*

(b)

Figura 3.25 Diagrama de bloques y tabla de verdad del ejemplo 3.23. **(a)** Diagrama de bloques. **(b)** Tabla de verdad.

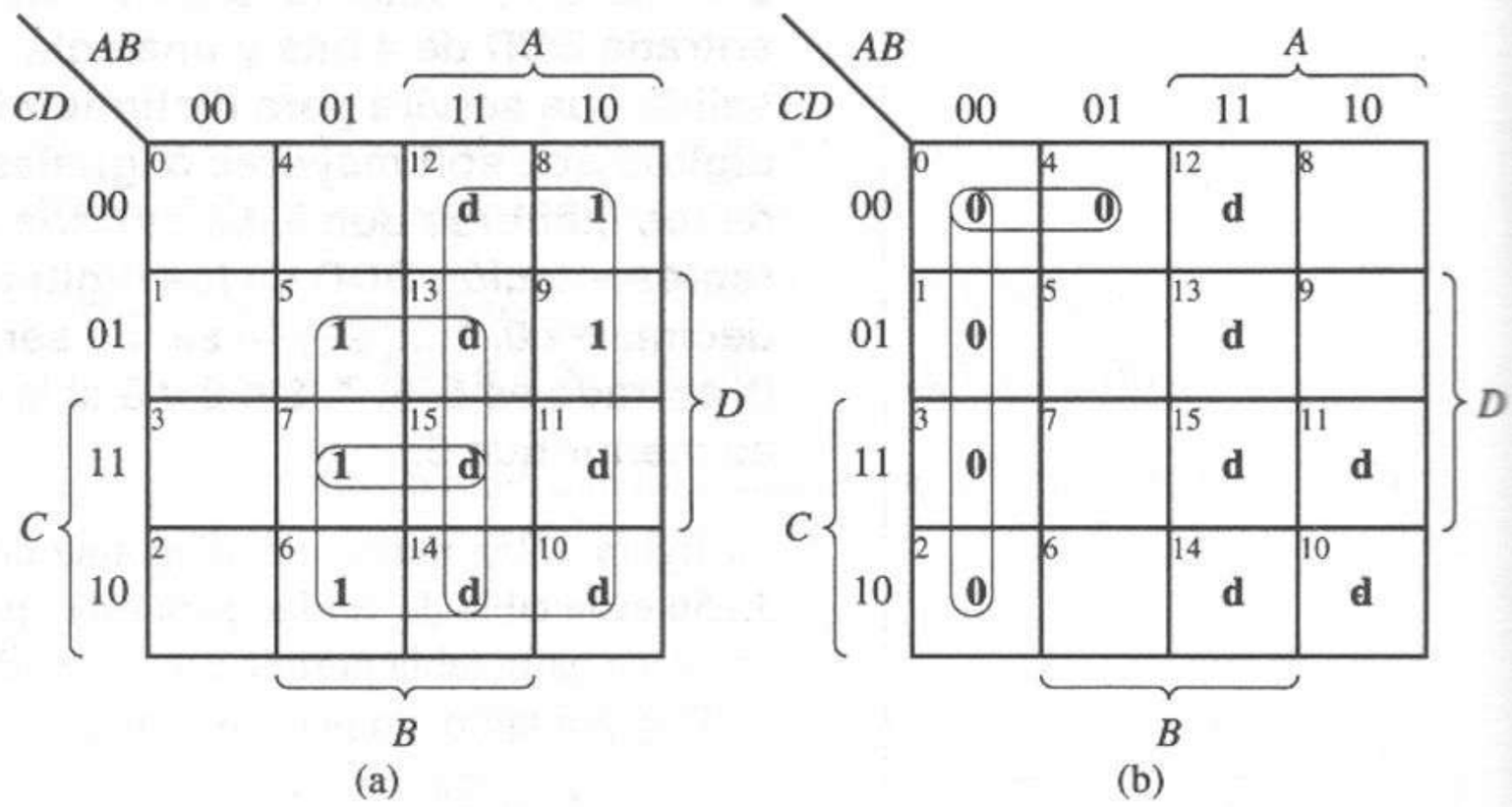


Figura 3.26 Uso de prescindibles para las formas SOP y POS. **(a)** Cubierta SOP mínima. **(b)** Cubierta POS mínima.

3.8 Uso de mapas K para eliminar riesgos de tiempo

Hasta ahora, hemos supuesto que el método óptimo para el diseño de cada circuito lógico combinatorio consiste en generar una expresión SOP o POS mínima y después realizarla con compuertas lógicas. Por desgracia, las cuestiones de tiempos exigen con frecuencia el uso de un circuito que no es mínimo.

Como vimos en el capítulo 2, cada compuerta lógica física tiene un tiempo de respuesta o retardo medible asociado a ella, dado por el tiempo que tarda en cambiar la salida de la compuerta después de un cambio en la entrada. Denotamos este tiempo de respuesta como T_{PHL} para un cambio de salida de alto a bajo y T_{PLH} para un cambio de bajo a alto.

El tiempo de respuesta de la mayor parte de los dispositivos lógicos es muy breve (desde nanosegundos hasta fracciones de un nanosegundo, según la tecnología, el *fan-in*, el *fan-out*, etcétera). Sin embargo, el tiempo de respuesta no puede ser exactamente el mismo para dos dispositivos, aun siendo del mismo tipo. Estas diferencias relativas en el tiempo de respuesta pueden provocar eventos no deseables en una red de conmutación. Estos eventos no deseables son los *riesgos* [3].

Como ejemplo de riesgo, consideremos la red de la figura 3.27a. Examinemos primero el caso ideal, suponiendo que las compuertas $G1$, $G2$ y $G3$ tienen tiempos de respuesta idénticos $T_{PHL} = T_{PLH} = \Delta t$. La figura 3.27c muestra el diagrama de tiempos del circuito para una secuencia particular de entradas. Observe que el cambio de x_3 en t_1 hace que y_2 cambie en el instante $t_2 = t_1 + \Delta t$, y que esto provoca un cambio en z en $t_3 = t_2 + \Delta t$. Un cambio de x_2 en t_4 no provoca cambios en las salidas de las compuertas. En t_5, el cambio de x_1 inicia los cambios de y_2 (de 1 a 0) y y_3 (de 0 a 1), ambos en $t_6 = t_5 + \Delta t$. Como $z = y_1 + y_2$, el resultado del cambio simultáneo de y_1 y y_2 no cambia a z.

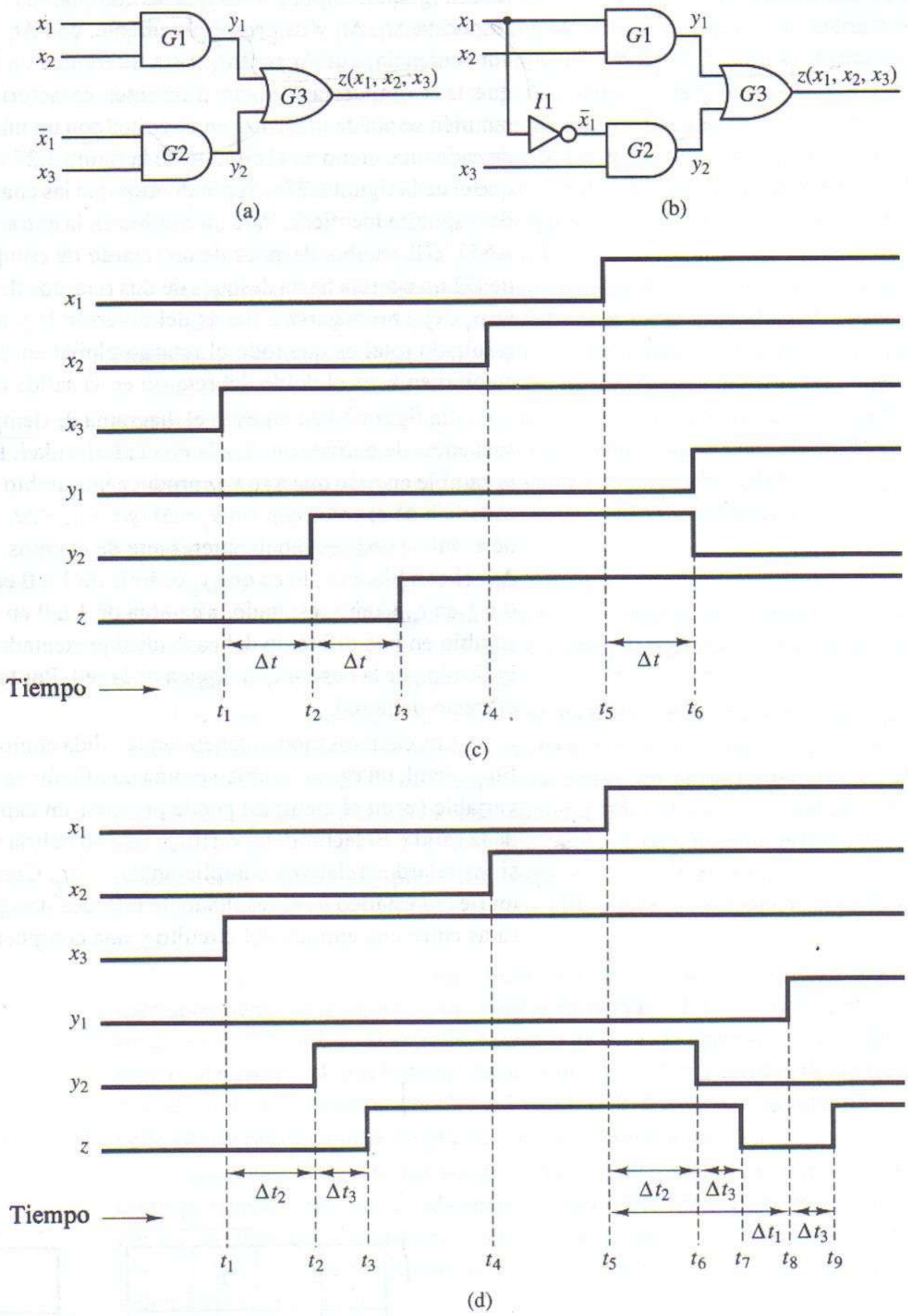


Figura 3.27 Ilustración de un riesgo estático. **(a)** Red con riesgo estático. **(b)** Red equivalente. **(c)** Diagrama de tiempos con retrasos idénticos Δt. **(d)** Diagrama de tiempos con retrasos $\Delta t_1 > \Delta t_2 > \Delta t_3$.

Ahora examinemos un caso más real, donde los retardos en las compuertas no son iguales. Supongamos que las compuertas $G1$, $G2$ y $G3$ tienen tiempos de respuesta Δt_1, Δt_2 y Δt_3, respectivamente, con $\Delta t_1 > \Delta t_2 > \Delta t_3$. Supongamos, por conveniencia, que $\Delta t_1 = 2\Delta t_2$. Esta diferencia en los retardos puede deberse al que las compuertas tengan diferentes características físicas. Esta situación también se puede presentar en circuitos con un número diferente de compuertas en cada ruta, como en el circuito de la figura 3.27b, que realiza la misma función que el de la figura 3.27a. Supondremos que las cuatro compuertas tienen tiempos de respuesta idénticos. Para un cambio en la entrada x_1, la salida de la compuerta AND, $G1$, cambia después de un retardo de compuerta, mientras que la salida de $G2$ no cambia hasta después de dos retardos de compuerta, ya que el cambio en x_1 debe propagarse a través del inversor $I1$ y de la compuerta AND, $G2$. El resultado total es que todo el retardo global en la salida de $G2$ después de un cambio en x_1 es el doble del retardo en la salida de $G1$.

La figura 3.27d muestra el diagrama de tiempos del circuito para la misma secuencia de entrada analizada con anterioridad. El cambio de x_3 en t_1 hace que y_2 cambie en t_2, lo que a su vez produce un cambio de z en t_3. Ésta es la secuencia esperada de eventos con $t_2 = t_1 + \Delta t_2$ y $t_3 = t_2 + \Delta t_3$. Sin embargo, en t_5 el cambio de x_1 inicia una secuencia interesante de eventos. En primer lugar, como $\Delta t_2 < \Delta t_1$, el cambio en x_1 hace que y_2 cambie de 1 a 0 en t_6 antes de que y_1 cambie de 0 a 1 en t_8. Como resultado, z cambia de 1 a 0 en t_7 y luego de 0 a 1 en t_9. Este cambio en z es diferente del caso ideal presentado en la figura 3.27b, y no está indicado por la descripción lógica de la red. Por tanto, no es el comportamiento correcto de la red.

Los cambios momentáneos en la salida como éste son los *riesgos estáticos*. En general, un riesgo estático es una condición en la cual un sencillo cambio de variable (x_1 en el ejemplo) puede producir un cambio momentáneo inesperado de la salida. El lector debe verificar que no habría riesgos en el ejemplo anterior si los retardos relativos cumplieran $\Delta t_1 \leq \Delta t_2$. Como se ilustra en este ejemplo, un riesgo estático es el resultado de retardos desiguales a lo largo de diferentes rutas entre una entrada del circuito y una compuerta de salida.

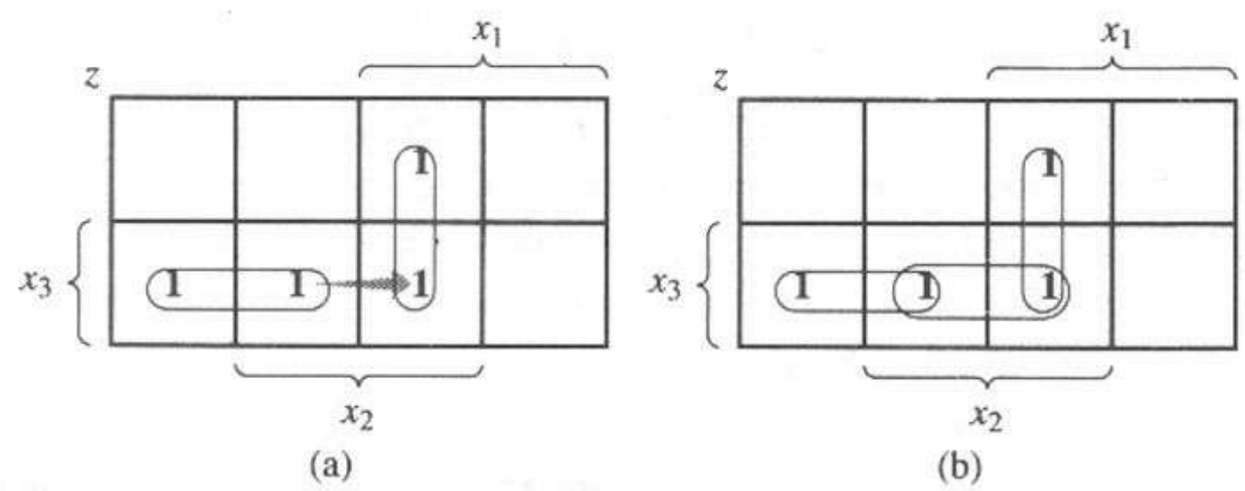


Figura 3.28 Identificación de riesgos en un mapa K. **(a)** Mapa K con condiciones de riesgo. **(b)** Mapa K con riesgos eliminados.

Podemos determinar la causa de la condición de riesgo anterior examinando el mapa K de la red de la figura 3.28a. En los circuitos correspondientes de las figuras 3.27a y 3.27b, obtenemos el término producto x_1x_2 de la compuerta AND $G1$ y el término producto $\overline{x}_1x_3$ de la compuerta AND $G2$. Como se ilustra en la figura 3.27c, los riesgos ocurren cuando un cambio en la entrada (de $x_1 = 0$, $x_2 = 1$, $x_3 = 1$ a $x_1 = x_2 = x_3 = 1$) requiere que los mintérminos correspondientes sean cubiertos por diferentes términos producto. En el circuito, esto significa que la salida de una compuerta AND cambia de 1 a 0 mientras que la salida de la otra compuerta AND cambia de 0 a 1. El riesgo se presenta cuando la salida de $G1$ va a 0 antes de que la salida de G_2 vaya a 1.

Podemos evitar los riesgos estáticos con un diseño lógico cuidadoso de modo que la salida sea independiente del orden en que cambian las señales. Podemos evitar los riesgos de la figura 3.27c agrupando los mintérminos como en la figura 3.28b, agregando un tercer término producto a la expresión suma de productos que utilizaríamos comúnmente. Presentamos el circuito resultante en la figura 3.29. En este circuito el término producto agregado, obtenido de la compuerta AND $G4$, permanece con el valor 1 mientras las salidas de $G1$ y $G2$ cambian. Por tanto, la salida de la compuerta OR $G3$ conserva el valor 1 sin importar que cambie primero, $G1$ o $G2$.

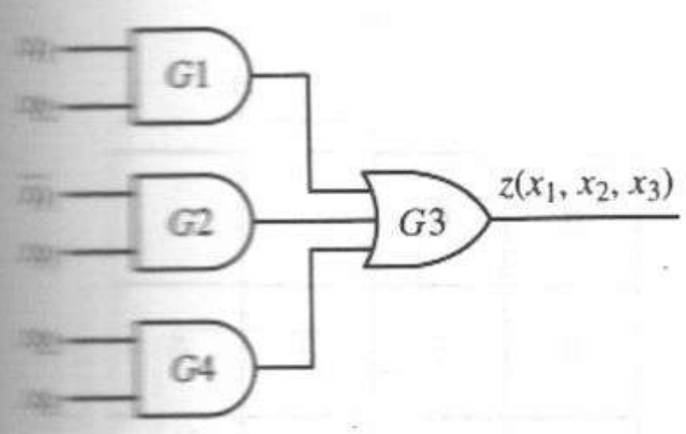


Figura 3.29 Red libre de riesgos.

En general, los riesgos se pueden eliminar cubriendo cada par de mintérminos adyacentes lógicamente con un término producto común. Por tanto, para eliminar riesgos necesitamos agregar compuertas redundantes a una red, obteniendo una realización no mínima.

Enfocamos el análisis anterior sobre los riesgos estáticos conocidos como *riesgos estáticos 1*, donde la salida debe conservar el valor lógico 1 pero cambia temporalmente a 0 lógico, con lo que obtenemos un pulso transitorio, o *falla*. *Los riesgos estáticos 0* también están contemplados en la definición de riesgos estáticos y pueden ocurrir. Los riesgos estáticos 1 se presentan principalmente en los circuitos AND–OR, que realizan expresiones SOP, mientras que los riesgos estáticos 0 se presentan en los circuitos OR–AND, que realizan expresiones POS.

Por ejemplo, consideremos el circuito de la figura 3.30a y su mapa K correspondiente de la figura 3.30b. Para la entrada $B = 1$, $C = 0$, $D = 1$, ocurre un riesgo estático 0 cuando la entrada A cambia de 0 a 1. Si observamos el diagrama lógico, el riesgo ocurre si la salida de la compuerta OR $G1$ cambia de 1 a 0 antes que la salida de la compuerta OR $G2$ cambie de 0 a 1. Así, aunque la salida debe conservar el valor 0, cambia de manera momentánea a 1.

Como en el caso de los riesgos estáticos 1, podemos evitar los riesgos estáticos 0 garantizando que cada par de maxtérminos adyacentes quede cubierto por un término suma. Las figuras 3.30c y d muestran una realización del circuito libre de riesgos y el mapa K correspondiente creado al agregar el término suma redundante $(\overline{B} + C + \overline{D})$.

Las redes también pueden presentar un segundo tipo de riesgo, conocido como *riesgo dinámico*. Un riesgo dinámico es una condición en la cual una salida debe cambiar de $0 \rightarrow 1$ o de $1 \rightarrow 0$ (es decir, el comportamiento de la salida será dinámico en vez de estático), pero cambia más de una vez antes de estabilizarse en su nuevo estado. Como en el caso de los riesgos estáticos, un riesgo dinámico también es causado por una condición especial de tiempo de respuesta

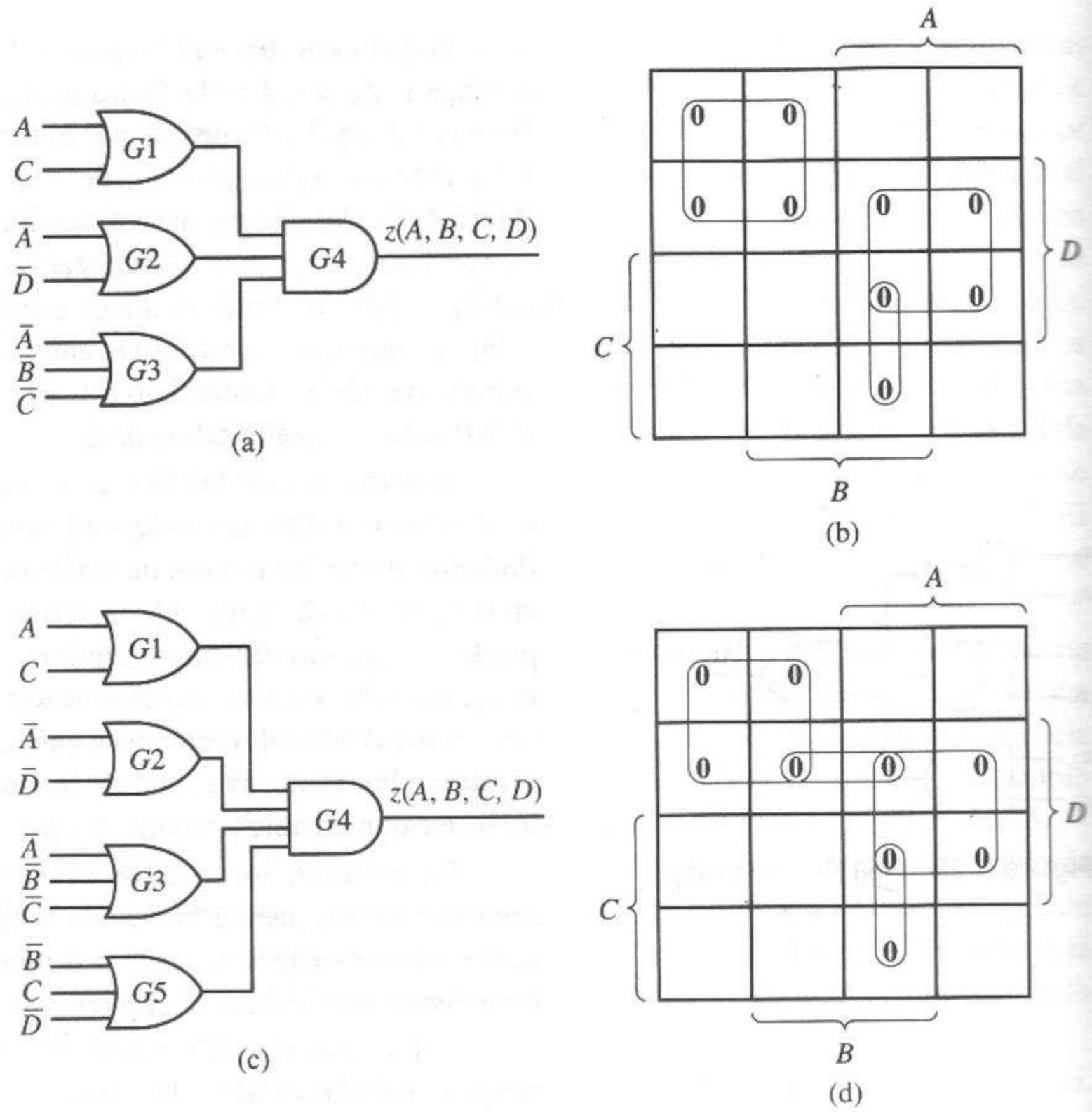


Figura 3.30 Ejemplo de riesgo estático 0. **(a)** Circuito con riesgo estático 0. **(b)** Mapa K con el riesgo. **(c)** Circuito libre de riesgos. **(d)** Mapa K con el riesgo eliminado.

Figura 3.31 Riesgos dinámicos. **(a)** Riesgo dinámico en un cambio 0 a 1. **(b)** Riesgo dinámico en un cambio 1 a 0.

relativa que ocurre después de una transición en la entrada que normalmente produce un cambio en la salida. Tales riesgos causan respuestas $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ para cambios normales $0 \rightarrow 1$, como se ilustra en la figura 3.31a, o $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ para cambios normales $1 \rightarrow 0$, como se muestra en la figura 3.31b. Se puede demostrar que los riesgos dinámicos son el resultado de riesgos estáticos existentes dentro del circuito. En consecuencia, las redes libres de riesgos estáticos también están libres de riesgos dinámicos. El lector debe consultar [4] si desea un análisis más profundo de la detección y eliminación de riesgos estáticos y dinámicos.

3.9 Método de minimización tabular de Quine-McCluskey

El *método de Quine-McCluskey (Q-M)* es un método tabular para la minimización de una función booleana [5, 6, 7]. Básicamente, el método Q-M tiene dos ventajas sobre el mapa K. La primera es que se trata de un método directo y sistemático para determinar una función mínima que depende menos de la habilidad del diseñador para reconocer patrones que el método de mapa K. La segunda es que el método es un esquema viable para el manejo de un gran número de variables, contrario a lo que ocurre en el mapa K, limitado en la práctica a cinco o seis variables. En general, el método Q-M realiza una búsqueda lineal ordenada sobre los mintérminos de la función para determinar todas las combinaciones de mintérminos adyacentes lógicamente. Como veremos, también podemos extender el método a funciones con varias salidas.

El método Quine-McCluskey parte de una lista de los mintérminos de n variables de la función y, de manera sucesiva, se deducen todos los implicantes con $n-1$ variables, los implicantes con $n-2$ variables, etcétera, hasta identificar todos los implicantes primos. Se deduce entonces una cubierta mínima de la función a partir del conjunto de implicantes primos. Ahora enumeraremos los cuatro pasos del proceso; los ejemplos subsecuentes ilustrarán el significado preciso de cada paso.

Paso 1. Enumerar en una columna todos los mintérminos de la función por minimizar, en su representación binaria. Separar los grupos según el número de bits 1 en su representación binaria. Esta partición facilita la identificación de los mintérminos adyacentes lógicamente ya que para serlo, dos mintérminos deben diferir exactamente en una literal y, por tanto, la representación binaria de un mintérmino debe tener un bit 1 menos o más que el otro.

Paso 2. Realizar una búsqueda exhaustiva de los mintérminos adyacentes entre grupos vecinos e incluirlos en una columna de implicantes de $(n-1)$ variables, marcando con $\surd$ cada mintérmino ya incluido. La representación binaria de cada nuevo implicante tiene un guión en la posición de la variable eliminada. Este procedimiento se repite para cada columna, combinando los implicantes de $(n-1)$ variables para obtener implicantes de $(n-2)$ variables, etcétera, hasta que no se puedan unir más implicantes. Cualquier término no eliminado representará un implicante primo de la función, pues no queda cubierto por un implicante mayor. El resultado final es una lista de implicantes primos de la función de conmutación.

Paso 3. Construir una tabla de implicantes primos que enumere los mintérminos en sentido horizontal y los implicantes primos en sentido vertical, escribiendo una entrada × cuando cierto implicante primo (fila) cubra un mintérmino (columna).

Paso 4. Seleccionar un número mínimo de implicantes primos que cubran a todos los mintérminos de la función de conmutación.

Presentaremos un ejemplo completo con estos cuatro pasos.

EJEMPLO 3.24

Utilizar la técnica Q-M para minimizar la función

$$f(A,B,C,D) = \sum m(2,4,6,8,9,10,12,13,15)$$

La figura 3.32 muestra el mapa K para este ejemplo; invitamos al lector a que intente obtener una función mínima mediante el método del mapa.

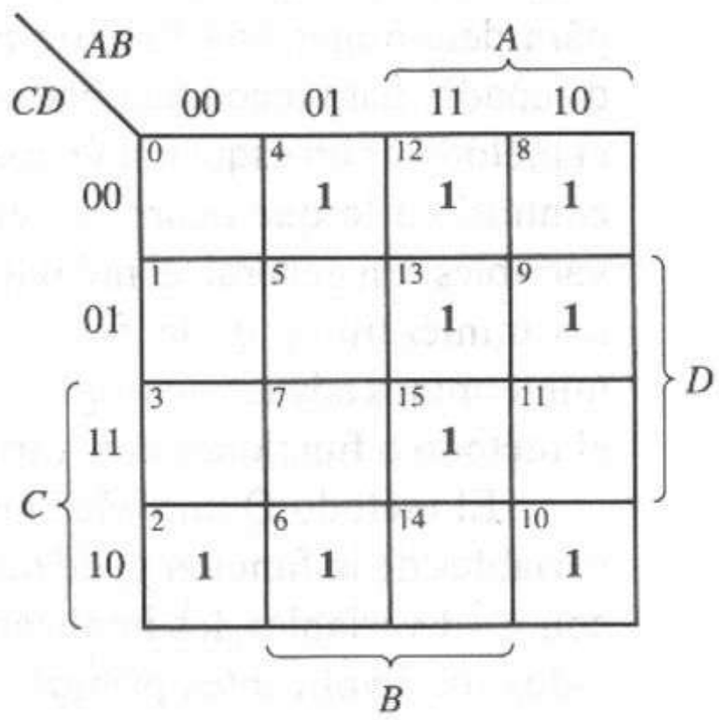


Figura 3.32 Mapa K para el ejemplo 3.30.

Paso 1. Para iniciar la técnica de minimización Q-M, agrupamos los mintérminos según el número de unos en la representación binaria del número de mintérminos. Esta agrupación de mintérminos se ilustra en la siguiente tabla:

Mintérminos	*ABCD*	
2	0010	
4	0100	Grupo 1 (un solo 1)
8	1000	
6	0110	
9	1001	Grupo 2 (dos unos)
10	1010	
12	1100	
13	1101	Grupo 3 (tres unos)
15	1111	Grupo 4 (cuatro unos)

Paso 2. Una vez obtenida esta tabla, iniciamos una búsqueda exhaustiva de todas las combinaciones de términos adyacentes lógicamente. Aquí resumiremos el método para realizar esta reducción funcional, que explicaremos con más detalle posteriormente. Consideremos la siguiente tabla de minimización, con las tres listas de mintérminos. Podemos combinar dos términos si y sólo si difieren en una sola literal. Por tanto, en la lista 1 sólo podemos combinar términos del grupo 1 con los del grupo 2. Cuando hayamos realizado todas las combinaciones entre estos dos grupos y las hayamos escrito en la lista 2, trazamos una recta bajo estas combinaciones, y comenzamos a combinar los términos del grupo 2 con los del grupo 3. Repetimos este sencillo procedimiento de una lista a otra para generar toda la tabla de minimización.

Lista 1			Lista 2			Lista 3		
Mintérminos	$ABCD$		Mintérminos	$ABCD$		Mintérminos	$ABCD$	
2	0010	√	2, 6	0-10	PI_2	8, 9, 12, 13	1-0-	PI_1
4	0100	√	2, 10	-010	PI_3			
8	1000	√	4, 6	01-0	PI_4			
6	0110	√	4, 12	-100	PI_5			
9	1001	√	8, 9	100-	√			
10	1010	√	8, 10	10-0	PI_6			
12	1100	√	8, 12	1-00	√			
13	1101	√	9, 13	1-01	√			
15	1111	√	12, 13	110-	√			
			13, 15	11-1	PI_7			

Debemos explicar algunos aspectos de la tabla. Observe que el primer elemento de la lista 2 indica que hemos combinado los mintérminos 2 y 6, pues sólo difieren en una literal. Los términos difieren en la variable B y, por tanto, aparece un guión en esa posición en la combinación 2, 6, lo que indica que hemos eliminado la variable B al combinar los dos mintérminos. Podemos comprobar fácilmente esta combinación mediante el álgebra booleana:

$$\text{mintérmino } 2 = \bar{A}\bar{B}C\bar{D}, \qquad \text{mintérmino } 6 = \bar{A}BC\bar{D}$$

y

$$\bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} = \bar{A}C\bar{D} \Rightarrow 0-10$$

Marcamos cada mintérmino de la lista 1 que se ha combinado con otro mediante una √, lo que indica que ha quedado incluido en un conjunto mayor. Aunque podemos combinar un término más de una vez, sólo lo marcamos una vez.

Una vez generada la lista 2 a partir de la lista 1, hacemos una búsqueda exhaustiva para combinar los términos de la lista 2 y así obtener la lista 3. Aquí resulta evidente por qué es importante marcar las variables que han sido eliminadas. Ya que, como antes, podemos combinar dos términos de la lista 2 sólo si difieren en una única literal, sólo podríamos combinar los términos a los cuales les falta la misma literal (un guión en la misma posición). Observe que en la lista 2 podemos combinar los mintérminos 8, 12 y 9, 13 y también 8, 9 y 12, 13 para obtener la combinación 8, 9, 12, 13 en la lista 3. Al analizar la lista 2, vemos que las combinaciones de mintérminos 8, 12 y 9, 13 carecen de la misma literal y difieren en otra. Lo mismo es cierto para la otra combinación. Por tanto, marcamos los cuatro términos de la lista 2 en la tabla. No podemos combinar otros términos de la lista 2; por tanto, los términos no marcados en toda la tabla son implicantes primos y los llamamos PI_1 ... PI_7. Ahora podríamos realizar la función como una suma de todos los implicantes primos; sin embargo, estamos buscando una realización mínima, por lo que sólo queremos utilizar el número mínimo necesario.

Una forma conveniente de verificar los errores en las listas 2, 3, 4, etcétera, es realizar la siguiente prueba en cada entrada: restamos los números de los mintérminos para verificar que hemos omitido las variables adecuadas. Por ejemplo, la entrada (4, 6 01-0) de la lista 2 indica que debemos eliminar la

variable con peso 6 – 4 = 2. En este ejemplo, los posibles pesos son 8, 4, 2 y 1. Para la entrada de la lista 3 (8, 9, 12, 13 ⇒ 1-0-):

$$9 - 8 = 1 \qquad 12 - 8 = 4$$

$$13 - 12 = 1 \qquad 13 - 9 = 4$$

de modo que debemos eliminar las variables con pesos 1 y 4.

Paso 3. Para determinar el mínimo número de implicantes primos necesarios para realizar la función, formamos una tabla de implicantes primos:

	2	4	6	√ 8	√ 9	10	√ 12	√ 13	√ 15
**PI$_1$				×	⊗		×	×	
PI$_2$	×		×						
PI$_3$	×					×			
PI$_4$		×	×						
PI$_5$		×					×		
PI$_6$				×		×			
**PI$_7$								×	⊗

Utilizamos la línea doble horizontal de la tabla entre PI$_1$ y PI$_2$ para separar los implicantes primos que contienen números diferentes de literales.

Paso 4. Al analizar las columnas de los mintérminos en la tabla de implicantes primos vemos que los mintérminos 9 y 15 quedan cubiertos cada uno por un sólo implicante primo (encerrado en un círculo). Por tanto, debemos elegir los implicantes primos 1 y 7, que entonces son implicantes primos esenciales (como lo indica el doble asterisco). Observe que al elegir estos dos implicantes primos, *también* hemos cubierto a los mintérminos 8, 12 y 13. Marcamos los cinco mintérminos cubiertos en la tabla con una √ sobre los números de los mintérminos.

El problema restante es seleccionar el menor número de implicantes primos (no esenciales) adicionales necesarios para cubrir los mintérminos 2, 4, 6 y 10. En general, realizamos esto formando una tabla reducida de implicantes primos, misma que mostramos a continuación. Observe que la tabla sólo contiene los mintérminos que faltan por cubrir y los candidatos a implicantes primos restantes para su inclusión en la cubierta.

	√ 2	√ 4	√ 6	√ 10
PI$_2$	×		×	
*PI$_3$	×			×
*PI$_4$		×	×	
PI$_5$		×		
PI$_6$				×

¿Cuál PI debemos seleccionar? Queda claro que los implicantes primos PI_5 y PI_6 son malas elecciones, pues sólo cubren un mintérmino, el cual también está cubierto por otro PI que cubre dos mintérminos. Observe que la mejor forma de cubrir los mintérminos 2, 4, 6 y 10 (con el número mínimo de implicantes primos) es elegir PI_3 y PI_4. El asterisco indica nuestra selección, y las marcas sobre los mintérminos restantes indican que hemos generado una cubierta completa. Por tanto, una realización mínima de la función original sería

$$\begin{aligned} f(A, B, C, D) &= PI_1 + PI_3 + PI_4 + PI_7 \\ &= 1\text{-}0\text{-} + \text{-}010 + 01\text{-}0 + 11\text{-}1 \\ &= A\bar{C} + \bar{B}C\bar{D} + \bar{A}B\bar{D} + ABD \end{aligned}$$

La figura 3.33 muestra los agrupamientos correspondientes de los mintérminos del mapa K.

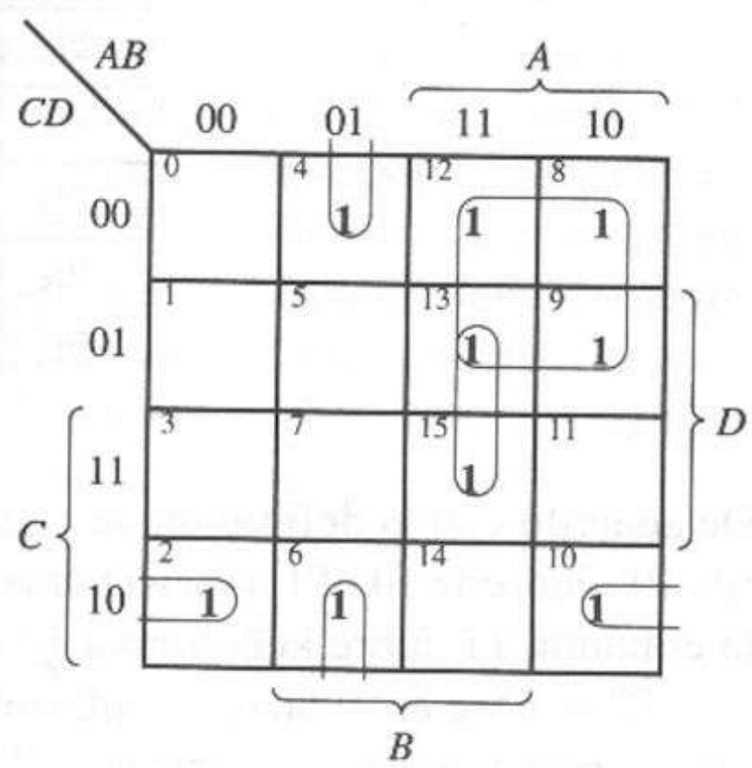


Figura 3.33 Agrupación de términos.

3.9.1 Procedimiento de cubierta

El problema de seleccionar un número mínimo de implicantes primos para realizar una función de conmutación se conoce también como *problema de cubierta*. Podemos emplear el siguiente procedimiento para elegir de manera sistemática un número mínimo de implicantes primos no esenciales a partir de la tabla de implicantes primos.

El primer paso consiste en eliminar de la tabla todas las filas de implicantes primos esenciales, al igual que las columnas de mintérminos cubiertos por ellos, como en el último ejemplo. Después simplificamos esta tabla reducida como sigue.

Una fila (columna) i de una tabla de implicantes primos *cubre* la fila (columna) j si la fila (columna) i contiene una × en cada columna (fila) donde j contiene una ×. Cada fila representa un implicante primo no esencial PI_i, mientras cada columna representa un mintérmino m_j de la función de conmutación. Por ejemplo, consideremos la siguiente tabla PI para la función de conmutación

$$f(A, B, C, D) = \sum m(0, 1, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15)$$

	√ 0	√ 1	5	√ 6	√ 7	√ 8	√ 9	10	11	13	√ 14	√ 15
** PI_1	⊗	×				×	×					
PI_2		×	×				×			×		
PI_3			×		×					×		×
PI_4						×	×	×	×			
PI_5							×		×	×		×
PI_6								×	×		×	×
** PI_7				⊗	×						×	×

Para esta tabla de implicantes primos, PI_1 y PI_7 son implicantes primos esenciales y están marcados con doble asterisco. Ahora eliminamos estas dos filas, al igual que todas las columnas en las que las filas tienen entradas ×. Obtenemos la siguiente tabla PI reducida:

	5	10	11	13
PI_2	×			×
PI_3	×			×
PI_4		×	×	
PI_5			×	×
PI_6		×	×	

De acuerdo con la definición de cubierta de filas y columnas ya establecida, la fila PI_2 cubre la fila PI_3 (y viceversa), la fila PI_4 cubre la fila PI_6 (y viceversa), la columna 11 cubre la columna 10 y la columna 13 cubre la columna 5.

Con base en el análisis anterior, podemos establecer las siguientes reglas para la reducción de las tablas de PI:

Regla 1. Una fila *cubierta* por otra fila se puede eliminar de la tabla. Si existen filas idénticas, se pueden eliminar todas las filas excepto una. En el ejemplo, podemos eliminar las filas PI_3 y PI_6.

Regla 2. Una columna que *cubre* otra columna se puede eliminar. Se pueden eliminar todas las columnas idénticas excepto una. En el ejemplo, podemos eliminar las columnas 11 y 13.

Si aplicamos estas reglas a la tabla de PI anterior, obtenemos la siguiente tabla de PI reducida:

	√ 5	√ 10
* PI_2	×	
* PI_4		×

Por tanto, podemos elegir PI_2 y PI_4 junto con los PI_1 y PI_7 esenciales para obtener una cubierta mínima de la función de conmutación.

Ahora analizaremos un tipo de tabla de PI que requiere un método especial para su reducción. Una tabla de PI *cíclica* es aquella que no tiene PI esenciales y que no se puede reducir mediante las reglas 1 y 2. A continuación mostramos una tabla cíclica para la función de conmutación

$$f(A, B, C) = \sum m(1, 2, 3, 4, 5, 6)$$

	√ 1	2	√ 3	4	5	6
*PI_1	×		×			
PI_2		×	×			
PI_3		×				×
PI_4				×		×
PI_5				×	×	
PI_6	×				×	

Verifique que ninguna fila o columna cubre otra fila o columna. El procedimiento para reducir una tabla cíclica consiste en seleccionar un PI de manera arbitraria en la tabla. Entonces, eliminamos de la tabla la fila correspondiente a este PI y las columnas correspondientes a los mintérminos cubiertos por el PI. Si la tabla reducida resultante no es cíclica, podremos aplicar las reglas 1 y 2. Sin embargo, si obtenemos otra tabla cíclica, repetimos el procedimiento para una tabla cíclica y de nuevo elegimos en forma arbitraria. Por ejemplo, elegimos PI_1 en la tabla cíclica anterior. Obtenemos la siguiente tabla no cíclica al eliminar la fila PI_1 y las columnas 1 y 3.

	2	4	5	6
PI_2	×			
PI_3	×			×
PI_4		×		×
PI_5		×	×	
PI_6			×	

Ahora podemos aplicar las reglas 1 y 2 para reducir esta tabla. PI_3 cubre la fila PI_2 por lo que podemos eliminar la fila PI_2. La fila PI_5 cubre la fila PI_6, por lo que podemos eliminar PI_6. La tabla reducida resultante es

	√ 2	√ 4	√ 5	√ 6
*PI_3	×			×
PI_4		×		×
*PI_5		×	×	

Así, hay que elegir PI_3 y PI_5 para cubrir la tabla.

Una cubierta mínima para la función de conmutación es PI_1, PI_3 y PI_5, aunque existen otras cubiertas mínimas. Podemos resumir el análisis anterior como sigue:

Paso 1. Identificar cualquier mintérmino cubierto por un único PI en la tabla. Seleccionar estos PI para la cubierta. Observe que este paso identifica los PI esenciales de entrada y los PI no esenciales en los pasos posteriores (a partir del paso 4).

Paso 2. Eliminar las filas correspondientes a los PI esenciales y no esenciales identificados. Eliminar las columnas correspondientes a los mintérminos cubiertos por las filas eliminadas.

Paso 3. Si se obtiene una tabla cíclica después del paso 2, ir al paso 5. En caso contrario, aplicar el procedimiento de reducción de las reglas 1 y 2.

Paso 4. Si se obtiene una tabla cíclica del paso 3, ir al paso 5. En caso contrario, regresar al paso 1.

Paso 5. Aplicar el procedimiento de la tabla cíclica. Repetir el paso 5 hasta obtener una tabla vacía o una tabla no cíclica. En el último caso, regresar al paso 1.

El procedimiento termina cuando el paso 2 o el 5 producen una tabla vacía, esto es, que no tiene filas ni columnas. Al aplicar por vez primera el paso 1, determinamos los implicantes primos que debemos incluir a fin de cubrir los mintérminos para los cuales sólo aparece una × en esa columna. Se identifican mediante un doble asterisco y son PI esenciales. En la segunda y siguientes aplicaciones del paso 1 (determinadas por el paso 4), identificamos los implicantes primos no esenciales mediante un asterisco a partir de las tablas de PI reducidas.

3.9.2 Funciones con especificación incompleta

La minimización de las funciones con términos prescindibles procede exactamente como en el ejemplo anterior, con una excepción importante, que ilustraremos en el siguiente ejemplo.

EJEMPLO 3.25

Utilizar el método Q-M para minimizar la función

$$f(A,B,C,D,E) = m(2,3,7,10,12,15,27) + d(5,18,19,21,23)$$

Seguimos el procedimiento del ejemplo anterior: enumeramos los mintérminos *y los prescindibles* en la tabla de minimización y los combinamos de la manera ya ilustrada. La siguiente tabla muestra los resultados de este procedimiento:

Lista 1			Lista 2			Lista 3		
Mintérminos	$ABCDE$		Mintérminos	$ABCDE$		Mintérminos	$ABCDE$	
2	00010	√	2, 3	0001-	√	2, 3, 18, 19	-001-	PI_1
3	00011	√	2, 10	0-010	PI_4	3, 7, 19, 23	-0-11	PI_2
5	00101	√	2, 18	-0010	√	5, 7, 21, 23	-01-1	PI_3
10	01010	√	3, 7	00-11	√			
12	01100	PI_7	3, 19	-0011	√			
18	10010	√	5, 7	001-1	√			
7	00111	√	5, 21	-0101	√			
19	10011	√	18, 19	1001-	√			
21	10101	√	7, 15	0-111	PI_5			
15	01111	√	7, 23	-0111	√			
23	10111	√	19, 23	10-11	√			
27	11011	√	19, 27	1-011	PI_6			
			21, 23	101-1	√			

Ahora debemos obtener una tabla de implicantes primos para este ejemplo. Es aquí donde el método difiere del ya descrito. Como algunos términos en la lista 1 son prescindibles, no hay necesidad de cubrirlos. Sólo hay que cubrir los mintérminos especificados, por lo que éstos son los únicos mintérminos que aparecen en la tabla de implicantes primos que se muestra a continuación. *No* enumere los prescindibles en la tabla de PI.

La tabla muestra que los implicantes primos esenciales son PI_4, PI_5, PI_6 y PI_7. Como el mintérmino 3 es el único que no queda cubierto por los implicantes primos esenciales, no es necesaria una tabla de implicantes primos reducida. Podemos cubrir el mintérmino 3 con PI_1 o PI_2, de modo que existen dos cubiertas mínimas para esta función. Las realizaciones mínimas para la función son

$$f(A, B, C, D, E) = PI_1 + PI_4 + PI_5 + PI_6 + PI_7$$

o bien

$$f(A, B, C, D, E) = PI_2 + PI_4 + PI_5 + PI_6 + PI_7$$

	√ 2	3	√ 7	√ 10	√ 12	√ 15	√ 27
PI_1	×	×					
PI_2		×	×				
PI_3			×				
** PI_4	×			⊗			
** PI_5			×			⊗	
** PI_6							⊗
** PI_7					⊗		

En términos de las variables,

$$f(A, B, C, D, E) = \bar{B}\bar{C}D + \bar{A}\bar{C}D\bar{E} + \bar{A}CDE + A\bar{C}DE + \bar{A}BC\bar{D}\bar{E}$$

o bien

$$f(A, B, C, D, E) = \bar{B}DE + \bar{A}\bar{C}D\bar{E} + \bar{A}CDE + A\bar{C}DE + \bar{A}BC\bar{D}\bar{E}$$

3.9.3 Sistemas con múltiples salidas

En el diseño de sistemas digitales, con frecuencia es necesario implantar más de una función de salida con algún conjunto dado de variables de entrada. Con las técnicas desarrolladas hasta ahora, podemos resolver el problema considerando cada función de manera individual. Sin embargo, podríamos compartir algunas compuertas y así obtener un diseño más sencillo y de mejor precio.

Realizamos la extensión del método tabular Q-M al caso de varias salidas igual que el caso singular, con las siguientes excepciones:

1. A cada mintérmino se le asocia una señal para identificar la función donde aparece.

2. Podemos combinar dos términos (o mintérminos) sólo si ambos tienen una o más señales comunes y el término resultante de las combinaciones sólo tiene señales comunes a ambos mintérminos.

3. Podemos eliminar cada término en la tabla de minimización sólo si todas las señales de éste aparecen en el término resultante de la combinación.

EJEMPLO 3.26

Utilizar el método tabular para obtener una realización mínima de las funciones

$$f_\alpha(A, B, C, D) = \sum m(0, 2, 7, 10) + d(12, 15)$$
$$f_\beta(A, B, C, D) = \sum m(2, 4, 5) + d(6, 7, 8, 10)$$
$$f_\gamma(A, B, C, D) = \sum m(2, 7, 8) + d(0, 5, 13)$$

Observe que este ejemplo ilustra también una minimización con prescindibles. Mostramos la tabla de minimización a continuación.

Min térms	Lista 1 $ABCD$	Señales		Min térms	Lista 2 $ABCD$	Señales		Min térm	Lista 3 $ABCD$	Señales	
0	0000	$\alpha\gamma$	√	0, 2	00-0	$\alpha\gamma$	PI_2	4, 5, 6, 7	01- -	β	PI_1
2	0010	$\alpha\beta\gamma$	PI_{10}	0, 8	-000	γ	PI_3				
4	0100	β	√	2, 6	0-10	β	PI_4				
8	1000	$\beta\gamma$	PI_{11}	2, 10	-010	$\alpha\beta$	PI_5				
5	0101	$\beta\gamma$	√	4, 5	010-	β	√				
6	0110	β	√	4, 6	01-0	β	√				
10	1010	$\alpha\beta$	√	8, 10	10-0	β	PI_6				
12	1100	α	PI_{12}	5, 7	01-1	$\beta\gamma$	PI_7				
7	0111	$\alpha\beta\gamma$	PI_{13}	5, 13	-101	γ	PI_8				
13	1101	γ	√	6, 7	011-	β	√				
15	1111	α	√	7, 15	-111	α	PI_9				

Consideremos la combinación 0, 8 de la lista 2. Este término es generado por la función $f_\gamma(A, B, C, D)$ a partir de los mintérminos 0 y 8 de la lista 1. No podemos eliminar el mintérmino 8, pues su etiqueta completa $\beta\gamma$ no está incluida en la etiqueta para el mintérmino 0. El mintérmino 0 está marcado gracias al término 0, 2 de la lista 2.

Es importante señalar que aunque hasta ahora nuestras tablas de minimización han tenido tres listas, en general el número de listas puede ser cualquier entero menor o igual a $n + 1$, donde n es el número de variables de entrada para la función de conmutación (o funciones, en el caso de varias salidas). A continuación mostramos la tabla de implicantes primos para la tabla de minimización (recuerde que *no* deben aparecer prescindibles en la parte superior):

	f_α				f_β			f_γ		
	√	√		√	√	√	√	√		
	0	2	7	10	2	4	5	2	7	8
** PI_1 β						⊗	×			
** PI_2 $\alpha\gamma$	⊗	×						×		
PI_3 γ										×
PI_4 β					×					
** PI_5 $\alpha\beta$		×		⊗	×					
PI_6 β										
PI_7 $\beta\gamma$							×		×	
PI_8 γ										
PI_9 α			×							
PI_{10} $\alpha\beta\gamma$		×			×			×		
PI_{11} $\beta\gamma$										×
PI_{12} α										
PI_{13} $\alpha\beta\gamma$			×						×	

La tabla muestra que PI_1, PI_2 y PI_5 son implicantes primos esenciales. A continuación mostramos la tabla de implicantes primos reducida; observe que hemos omitido los implicantes primos que sólo cubren prescindibles.

	f_α	f_γ	
	√	√	√
	7	7	8
* PI_3 γ			×
PI_7 $\beta\gamma$		×	
PI_9 α	×		
PI_{11} $\beta\gamma$			×
* PI_{13} $\alpha\beta\gamma$	×	×	

Es claro que el mejor conjunto de implicantes primos restantes es PI_3 y PI_{13}. Elegimos PI_3 en vez de PI_{11} porque tiene menos literales. Por tanto, las realizaciones mínimas para las tres funciones son

$$f_\alpha = PI_2 + PI_5 + PI_{13}$$
$$f_\beta = PI_1 + PI_5$$
$$f_\gamma = PI_2 + PI_3 + PI_{13}$$

o bien

$$f_\alpha = \bar{A}\bar{B}\bar{D} + \bar{B}C\bar{D} + \bar{A}BCD$$
$$f_\beta = \bar{A}B + \bar{B}C\bar{D}$$
$$f_\gamma = \bar{A}\bar{B}\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}BCD$$

Es importante observar que PI_2, PI_5 y PI_{13} sólo se generan una vez, pero que se utilizan para implantar dos de las funciones, como se muestra en la figura 3.34.

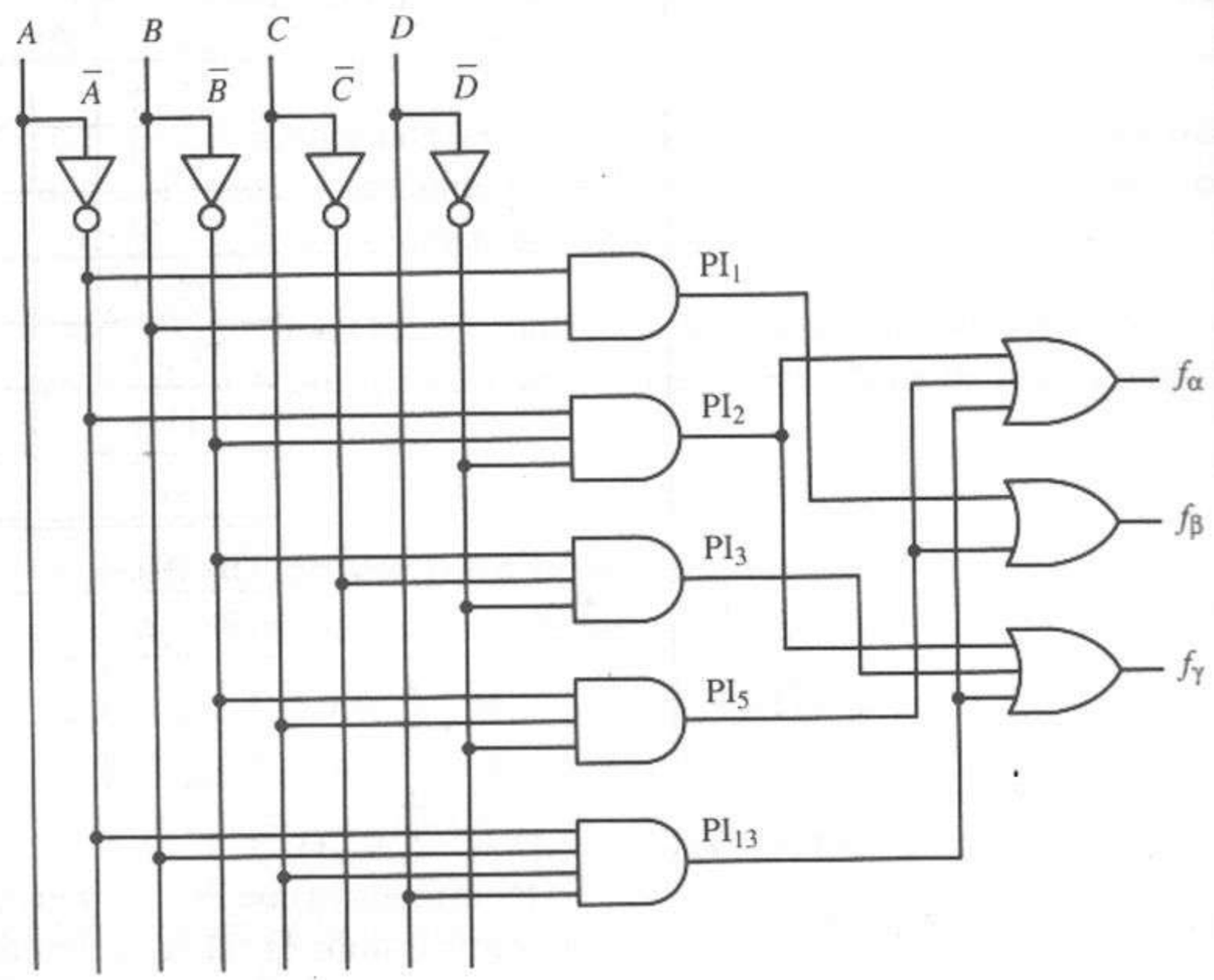


Figura 3.34 Circuito con varias salidas reducido.

3.10 Algoritmo de Petrick

Como ya habíamos señalado, los métodos presentados en la sección anterior para seleccionar una cubierta mínima son heurísticos y, por tanto, no garantizan la obtención de una solución óptima. En particular, los últimos pasos de los algoritmos 3.2 y 3.4 del mapa K y el último paso del método de Quine-McCluskey se basan en la heurística y el talento del diseñador para identificar un conjunto de implicantes primos mínimo para completar una cubierta, después de determinar los implicantes primos esenciales. Con frecuencia, se utiliza un método de prueba y error para identificar y evaluar las diversas cubiertas posibles.

Se han inventado varios métodos que pueden generar soluciones óptimas de manera directa, con un cierto costo en términos de tiempo adicional de cómputo. Uno de estos métodos es el *algoritmo de Petrick* [8], el cual utiliza un método algebraico para generar todas las cubiertas posibles de una función. El siguiente algoritmo para determinar cubiertas mínimas se basa en el algoritmo de Petrick.

Algoritmo 3.6

1. Utilizar el algoritmo 3.2, el algoritmo 3.4 o el método de Quine-McCluskey para determinar todos los implicantes primos de la función.
2. Crear una tabla de implicantes primos y después identificar y eliminar todos los implicantes primos esenciales, como en el paso 3 del método de Quine-McCluskey.
3. Para la tabla restante, escribir una expresión POS lógica que represente todas las cubiertas posibles, como sigue:
   a. Para cada columna de mintérmino m_i, escribir una expresión para la suma (OR) de todos los implicantes primos que cubren a m_i. Esta expresión indica que se puede elegir cualquiera o varios de estos implicantes primos para cubrir a m_i.
   b. Formar el producto (AND) de todos los términos suma del paso a. Este producto indica que todos los mintérminos deben quedar cubiertos.
4. Convertir la expresión lógica del paso 2 del formato POS al SOP mediante la ley distributiva, y después simplificar la expresión usando involución y absorción para eliminar los términos redundantes. Cada término producto de la expresión SOP resultante representa una posible cubierta.
5. Seleccionar la cubierta con el menor costo; en este caso, el "costo" se calcula como el número de implicantes primos en el término producto y el número de literales en cada implicante primo.

Utilizaremos los ejemplos de la sección anterior para ilustrar el algoritmo de Petrick.

EJEMPLO 3.27

Utilizar el algoritmo 3.6 para determinar una cubierta mínima para la función del ejemplo 3.24.

Después de formar la tabla de implicantes primos y eliminar los PI esenciales, obtenemos la siguiente tabla de PI reducida.

	√ 2	√ 4	√ 6	√ 10
PI_2	×		×	
*PI_3	×			×
*PI_4		×	×	
PI_5		×		
PI_6				×

Vemos que podemos cubrir m_2 con PI_2 o PI_3, el mintérmino m_4 con PI_4 o PI_5, el mintérmino m_6 con PI_2 o PI_4 y el mintérmino m_{10} con PI_3 o PI_6. La expresión SOP C que representa todas las cubiertas de estos mintérminos es

$$C = (PI_2 + PI_3)(PI_4 + PI_5)(PI_2 + PI_4)(PI_3 + PI_6)$$

Al convertir a la forma SOP mínima, obtenemos

$$C = PI_2PI_3PI_5 + PI_3PI_4 + PI_2PI_4PI_6 + PI_2PI_5PI_6$$

Por esta expresión, vemos que existen cuatro cubiertas no redundantes de los cuatro mintérminos restantes. De éstas, la cubierta que comprende a PI_3 y PI_4 representa la de costo mínimo, pues sólo consta de dos términos producto, mientras que las otras tres cubiertas requieren tres términos cada una.

El siguiente ejemplo muestra la utilidad del algoritmo de Petrick al trabajar con tablas PI que no se pueden reducir por el dominio de las filas o las columnas.

EJEMPLO 3.28

Utilizar el algoritmo 3.6 para determinar una cubierta mínima para la función $f(A, B, C) = \sum m(1, 2, 3, 4, 5, 6)$ a partir de la siguiente tabla de PI.

	√ 1	2	√ 3	4	5	6
*PI_1	×		×			
PI_2		×	×			
PI_3		×				×
PI_4				×		×
PI_5				×	×	
PI_6	×				×	

Observe que la tabla es cíclica; es decir, no hay PI esenciales. Además, todos los PI contienen el mismo número de literales, de modo que no existe una primera elección obvia de una cubierta. Aplicando el algoritmo 3.6, la forma POS de todas las cubiertas, C, convertida a la forma SOP mínima es

$$\begin{aligned} C &= (PI_1 + PI_6)(PI_2 + PI_3)(PI_1 + PI_2)(PI_4 + PI_5)(PI_5 + PI_6)(PI_3 + PI_4) \\ &= PI_1PI_3PI_5 + PI_2PI_3PI_5PI_6 + PI_1PI_2PI_4PI_5 + PI_2PI_4PI_6 + PI_2PI_3PI_5PI_6 \end{aligned}$$

De esta expresión deducimos que existen cinco cubiertas no redundantes. Tres de ellas contienen cuatro implicantes primos, mientras que las cubiertas $PI_1PI_3PI_5$ y $PI_2PI_4PI_6$ sólo contienen tres. Para esta función, todos los implicantes primos contienen el mismo número de literales. Por tanto, $PI_1PI_3PI_5$ y $PI_2PI_4PI_6$ representan soluciones mínimas.

Aunque el algoritmo de Petrick identifica todas las cubiertas no redundantes posibles y, por tanto, permite identificar una solución óptima, su complejidad aumenta bastante con el número de mintérminos y de implicantes primos, por lo que no es práctico para funciones de gran tamaño. En la siguiente sección

analizaremos los métodos asistidos por computadora que podemos aplicar a funciones de tamaño arbitrario.

3.11 Mínimización de funciones de conmutación asistida por computadora

Aunque el mapa de Karnaugh es una de las herramientas más eficaces para minimizar manualmente las funciones de conmutación cuando el número de variables n es pequeño, se vuelve poco práctico para n mayor que 4 o 5. Al ser una técnica gráfica, el manejo de los mapas K no se puede implantar con facilidad en programas de cómputo.

Los métodos de minimización de circuitos de conmutación que manejan tablas, como el método de Quine-McCluskey, o los métodos que trabajan de manera directa con las expresiones de conmutación, se pueden extender con facilidad a funciones con una cantidad arbitraria de variables. Muchos de estos métodos se pueden automatizar mediante su programación en computadoras digitales. En consecuencia, utilizamos los métodos tabulares y algebraicos en vez de los mapas K para el trabajo práctico de diseño; aquellos métodos forman la base de los algoritmos para la minimización de funciones de conmutación incorporados a los sistemas de diseño asistido por computadora (CAD). Como ya hemos dicho en relación con los métodos de mapa K y Quine-McCluskey, y como se muestra en la figura 3.35, seguimos cuatro pasos básicos para deducir una expresión de conmutación mínima en forma de suma de productos para una función lógica.

1. Representar la función con un formato adecuado para el método por utilizar.
2. Determinar el conjunto completo de implicantes primos de la función.
3. Determinar los implicantes primos esenciales a partir del conjunto de implicantes primos.
4. Seleccionar los implicantes primos no esenciales necesarios para completar una cubierta mínima de la función.

Para las funciones con especificación incompleta, consideramos los términos prescindibles como términos 1 en los pasos 1 y 2, para que los implicantes primos sean lo más grandes que se pueda, y después los ignoramos en los pasos 3 y 4, pues no tienen que ser cubiertos al realizar la función.

La comparación de los diversos algoritmos para minimización de las funciones de conmutación muestra una relación estrecha entre el uso de la memoria, el tiempo de ejecución y la optimalidad de los resultados [9, 10]. Por esta razón, muchos sistemas de CAD ofrecen diversos algoritmos para que el diseñador elija el que aproveche de mejor manera las características de la función por minimizar.

Entre los factores que influyen en el uso de la memoria y el tiempo de cómputo están el número de mintérminos de la función, la cantidad de implicantes primos, el número de mintérminos no cubiertos por los implicantes primos esenciales de la función y el número de salidas del circuito, es decir, el número de funciones que pueden minimizarse de manera simultánea. La propia naturaleza del algoritmo influye sobre el tiempo de cómputo, incluyendo el método de representación de los términos, la eficiencia de cada paso individual, el orden de los términos, y el grado en que las hipótesis establecidas al desarrollar la heurística del algoritmo se ajustan a la función en cuestión.

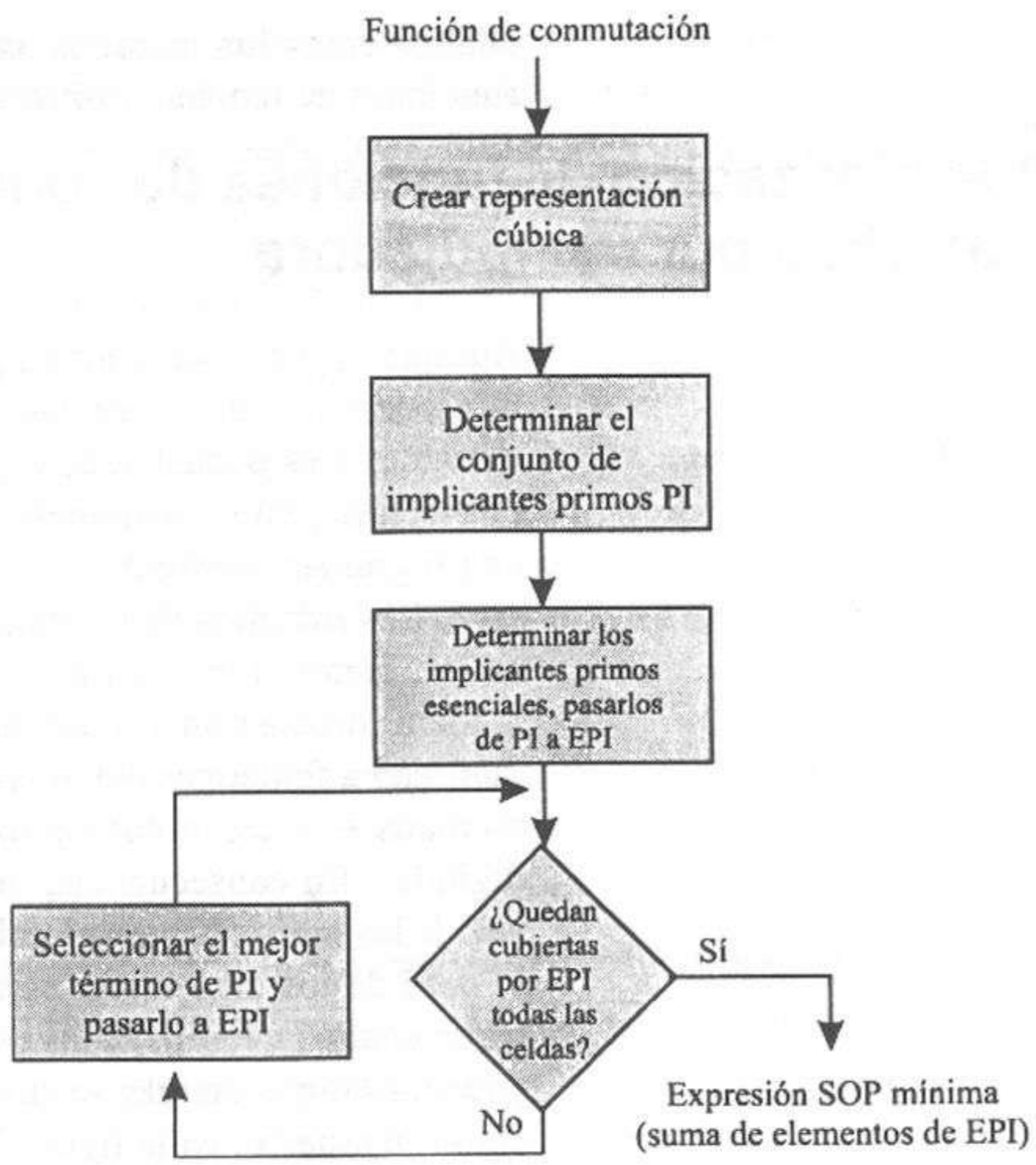


Figura 3.35 Minimización de las funciones de conmutación.

Por ejemplo, los requisitos de cómputo y espacio de almacenamiento de los algoritmos tabulares de minimización crecen con rapidez cuando aumenta el número de entradas de funciones o de mintérminos. El método de Quine-McCluskey requiere la generación de la lista completa de mintérminos de la función por minimizar. (Observe que el total de mintérminos posibles puede ser hasta 2^n para una función con n entradas.) Todos estos términos se deben guardar en forma individual en la memoria de la computadora, aunque la función se pueda describir mediante una cantidad pequeña de términos producto o suma. Entonces, para determinar los implicantes primos, hay que comparar todos los mintérminos por pares para obtener 1-cubos, hay que comparar todos los pares de 1-cubos para formar 2-cubos, etcétera. Por tanto, el número de operaciones de comparación podría ser demasiado grande si el número de mintérminos también lo es. Mostraremos esto en el siguiente ejemplo.

EJEMPLO 3.29

Determinar la cantidad de mintérminos de la función

$$f(a,b,c,d,e,f) = a + abcdef$$

así como el número de operaciones de comparación necesarias para reducir la expresión mediante el método de Quine-McCluskey.

El desarrollo de la función produce 32 mintérminos, que combinamos mediante el método de Quine-McCluskey en dieciséis 1-cubos, ocho 2-cubos, cuatro 3-cubos, dos 4-cubos y un 5-cubo, para un total de 63 por guardar. En total, debemos realizar 651 operaciones de comparación para deducir el 5-cubo a.

Si usamos el álgebra de conmutación, podemos reducir la función con una aplicación del teorema 4 (absorción):

$$f(a, b, c, d, e, f) = a + abcdef = a$$

Este ejemplo muestra que el manejo algebraico de los términos producto originales de una expresión de conmutación podría ser preferible a la aplicación del método de Quine-McCluskey, en particular cuando el número de términos es pequeño.

En la siguiente sección examinaremos algunos métodos algebraicos utilizados en los sistemas de CAD para automatizar los pasos del proceso de minimización. Ésta no es una lista exhaustiva de técnicas, sino que pretende proporcionar al lector una idea general de la naturaleza de los algoritmos utilizados. El lector debe consultar [9, 10] si desea un análisis más a fondo y ejemplos de programas de cómputo que implantan varios de los algoritmos descritos en este capítulo.

3.11.1 Representación cúbica de las funciones de conmutación

La notación más frecuente en los sistemas de CAD para la representación de las expresiones de conmutación durante la captura e impresión del diseño, así como para uso interno, es la notación *cúbica*. Un cubo geométrico n-dimensional (n-*cubo*, para abreviar) tiene 2^n vértices, cada uno de los cuales está unido a otros n vértices mediante líneas (aristas) a lo largo de las n dimensiones de la figura. Cada vértice se identifica de manera única mediante un número binario de n bits, representado por una n-ada $x_1x_2...x_n$. Cada pareja de vértices adyacentes de un n-cubo (los vértices unidos en forma directa mediante una arista) difieren exactamente en una posición de bit, x_i, correspondiente a la i-ésima dimensión a lo largo de la cual se encuentra la arista que los une. La figura 3.36a muestra esto para un 3-cubo.

Podemos representar una función de conmutación de n variables, $f(x_1,...,x_n)$ mediante una asociación de sus mintérminos con los vértices de un n-cubo. Esta asociación utiliza la codificación binaria de los mintérminos descrita en el capítulo 2, donde $x_i = 0$ indica que x_i está complementado en el mintérmino y $x_i = 1$ indica una variable no complementada. Por ejemplo, sea

$$f_\delta(a, b, c) = \sum m(2, 4, 6) = \bar{a}b\bar{c} + a\bar{b}\bar{c} + ab\bar{c} = \{010, 100, 110\}$$

Asociamos estos tres mintérminos a los tres vértices 010, 100 y 110 en un 3-cubo, respectivamente, como en la figura 3.36b. Cada mintérmino aparece como un vértice (un punto más grueso) en la figura.

Podemos representar un implicante con r variables de una función de n variables, formado al combinar mintérminos adyacentes, mediante un subcubo (o r-cubo) del n-cubo. Por ejemplo, en la figura 3.36, la arista (1-cubo) que une los vértices 010 y 110 corresponde al implicante $b\bar{c}$, pues, por el teorema 6(a),

$$\bar{a}b\bar{c} + ab\bar{c} = b\bar{c}$$

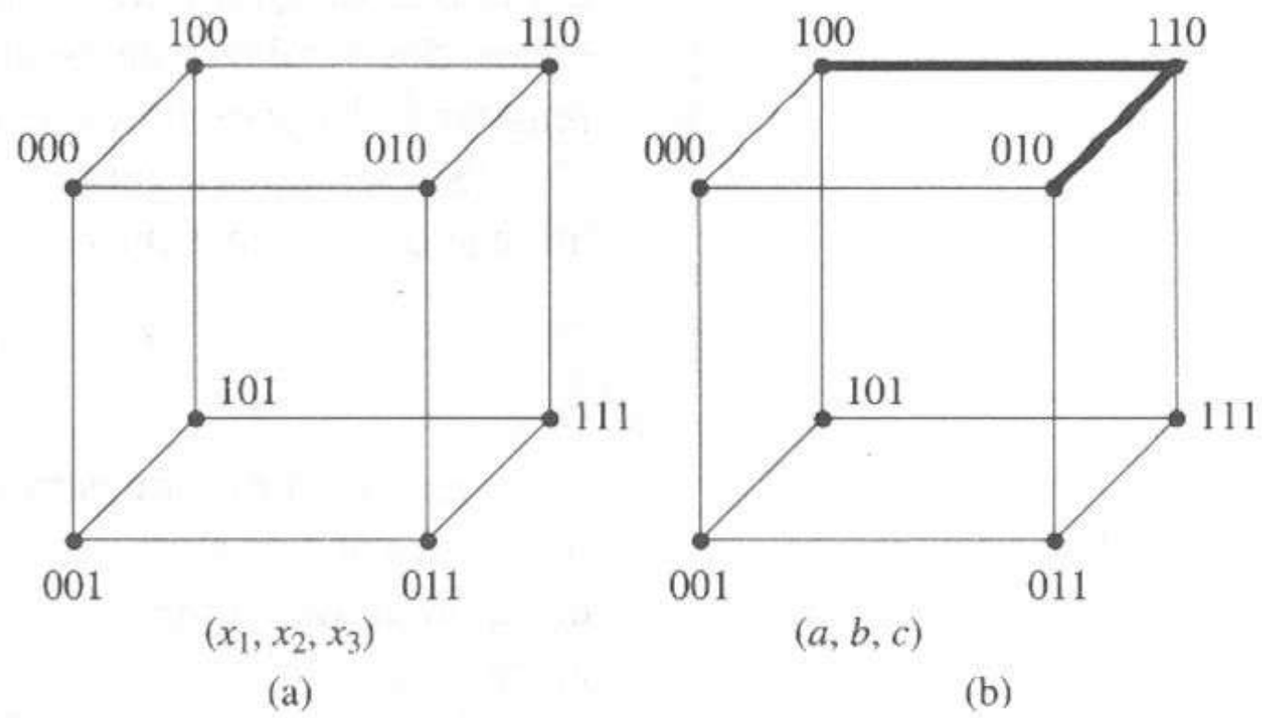


Figura 3.36 Asociación de las funciones de conmutación con cubos *n*-dimensionales. **(a)** 3-cubo. **(b)** $f_\delta(a,b,c) = \Sigma\, m(2, 4, 6)$ asociada a un 3-cubo.

Representamos los implicantes en la notación cúbica mediante el guión (–) en vez de cada variable eliminada. Por ejemplo, representamos el implicante $b\bar{c}$ de $f_\delta(a, b, c)$ como –10. En general, formamos un *r*-cubo de una función con *n* variables combinando 2^r términos adyacentes, lo que elimina *r* variables. Así, un *r*-cubo tiene *r* símbolos –. Podemos expresar una forma suma de productos de una función con *n* variables $f(x_1,...,x_n)$ mediante una lista de cubos, cada uno de los cuales representa un término producto.

EJEMPLO 3.30

Determinar la representación cúbica de

$f(x,y,z) = y + \bar{x}z.$

$$\begin{aligned} y &\Rightarrow -1- \\ \bar{x}z &\Rightarrow 0-1 \end{aligned}$$

Por tanto, podemos representar la función mediante el conjunto de cubos {–1–, 0–1}.

Al implantar las herramientas de minimización y simulación lógica en una computadora digital, con frecuencia se utilizan variantes de la representación cúbica básica para que los programadores utilicen las estructuras de datos y operaciones inherentes del lenguaje de programación particular que implanta las herramientas. Muchos de estos programas permiten introducir o imprimir las expresiones de conmutación como expresiones algebraicas, listas de mintérminos, etcétera, para después convertirlas a la representación cúbica para su uso interno. [9, 10] contienen algunos ejemplos.

3.11.2 Métodos algebraicos para determinar implicantes primos

Después de representar una función de conmutación de manera adecuada, el proceso de minimización se inicia determinando los implicantes primos de la función. En un mapa K, esto implica identificar y encerrar en un círculo el mayor

grupo de celdas que cubran a cada mintérmino del mapa. En el método de Quine-McCluskey, se comparan las pares de cubos, y los cubos que difieren precisamente en una literal se combinan para crear cubos de mayor tamaño hasta que ya no puedan realizarse más combinaciones.

Los métodos algebraicos aplican ciertos teoremas del álgebra de conmutación a las listas de términos producto. Estos algoritmos minimizan los requisitos de memoria trabajando en forma directa con los implicantes dados (términos producto) representados mediante la notación cúbica, en vez de pedir un desarrollo de la función a una lista de mintérminos. En el caso de funciones descritas por una cantidad pequeña de implicantes, la eficiencia de cómputo se incrementa, aunque el número de variables de entrada pueda ser grande, pues el número de operaciones se minimiza.

La mayor parte de los algoritmos algebraicos se basan en el teorema de consenso del álgebra booleana (teorema 9) y se implantan con facilidad en los paquetes de diseño asistido por computadora. En estos algoritmos se utiliza el teorema de consenso para combinar los implicantes adyacentes en otros mayores e identificar nuevos implicantes que se traslapan con los que ya estén en la lista. El siguiente ejemplo ilustra estas dos operaciones.

EJEMPLO 3.31

Utilizar el teorema de consenso para simplificar la función

$$f(A,B,C,D) = \bar{A}BD + ABD + \bar{A}\bar{B}\bar{C}$$

La función, que se indica en la figura 3.37a, queda cubierta por el conjunto de tres implicantes $\{\bar{A}BD, ABD, \bar{A}\bar{B}\bar{C}\}$. Comenzaremos aplicando el teorema de consenso a los dos primeros términos de la lista.

$$\begin{aligned} \bar{A}BD + ABD &= \bar{A}BD + ABD + (BD)(BD) && \text{[T9(a), consenso]} \\ &= \bar{A}BD + ABD + BD && \text{[T1(b), idempotencia]} \\ &= BD && \text{[T4(a), absorción]} \end{aligned}$$

El término de consenso BD cubre (absorbe) a $\bar{A}BD$ y ABD y, por tanto, puede reemplazar a ambos en el conjunto de implicantes. Observe que esto equivale a la combinación de los dos implicantes adyacentes $\bar{A}BD$ y ABD por el teorema 6(a).

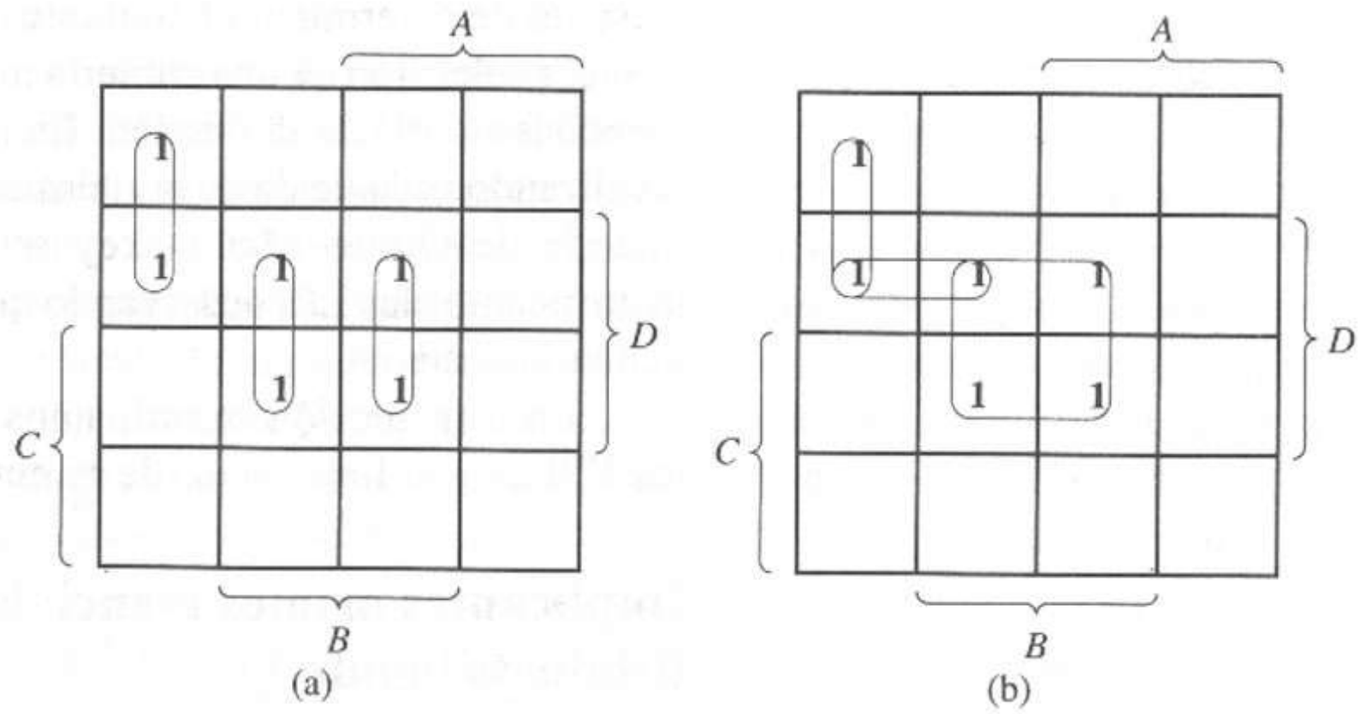


Figura 3.37 Determinación de los implicantes primos de $f(A, B, C, D) = \bar{A}BD + ABD + \bar{A}\bar{B}\bar{C}$. **(a)** Términos dados. **(b)** Implicantes primos.

Ahora, aplicamos el teorema de consenso a los otros dos términos de la lista de implicantes para obtener

$$\bar{A}\bar{B}\bar{C} + BD = \bar{A}\bar{B}\bar{C} + BD + \bar{A}\bar{C}D \quad \text{[T9(a), consenso]}$$

Como lo muestra la figura 3.37b, el término de consenso $\bar{A}\bar{C}D$ no cubre $\bar{A}\bar{B}\bar{C}$ ni a BD por completo. Por tanto, $\bar{A}\bar{C}D$ representa otro implicante primo de la función y se agrega a la lista.

La figura 3.37b muestra la lista final de implicantes primos, $\{\bar{A}\bar{B}\bar{C}, BD, \bar{A}\bar{C}D\}$.

La aplicación de los teoremas de consenso, idempotencia y absorción a los términos representados en notación cúbica se automatiza con facilidad mediante operadores especiales. El más común es el *producto STAR*, descrito en [11], operador que identifica las posiciones de bit en que difieren dos cubos.

En el *algoritmo de consenso iterativo* [10, 11], se obtiene una lista completa de implicantes primos para una función partiendo de una lista de cubos que describe a la función, aplicando el teorema de consenso a todos los pares de cubos de la lista, utilizando el producto STAR, e identificando los cubos de consenso, como en los ejemplos anteriores. Cada cubo de consenso se agrega a la lista si no ha sido cubierto (absorbido) por un cubo que ya esté en la lista, y todos los cubos cubiertos (absorbidos) por un cubo de consenso recién agregado se eliminan de la lista. Este proceso de agregar un nuevo cubo a la lista se conoce como formación de la *unión absorbida* de la lista y el cubo. Después de comparar todos los pares de cubos de la lista, sólo quedan los que representan a los implicantes primos.

El número de aplicaciones del producto STAR en el algoritmo de consenso iterativo es grande si el número de términos producto es grande. El *algoritmo de consenso generalizado* [10, 12] utiliza el mismo principio básico, pero reduce la cantidad total de operaciones al organizar en forma sistemática los términos antes de aplicar el operador STAR, de manera similar a la forma en que agrupamos mintérminos en el paso 1 del método de Quine-McCluskey para facilitar la identificación de los términos adyacentes lógicamente.

3.11.3 Identificación de implicantes primos esenciales

Después de determinar el conjunto de implicantes primos PI de una función, se inicia la selección de una cubierta mínima, identificando los implicantes primos esenciales (EPI) de la función. En un mapa K, los EPI se destacan fácilmente, localizando cada celda de mintérmino en el mapa encerrada en un solo PI. En el método de Quine–McCluskey se utiliza una tabla de implicantes primos; identificamos los EPI observando que columnas de mintérminos de la tabla sólo tienen una marca.

En esta sección examinamos dos métodos algebraicos para determinar los EPI de una función desde su conjunto completo de implicantes primos.

Implicantes primos esenciales mediante el producto agudo

Dado un conjunto PI de implicantes primos de una función representada en notación cúbica, queremos saber si un implicante primo dado cubre uno o más

mintérminos de la función que no estén cubiertos por los demás implicantes primos de PI. Podemos determinar esto mediante un operador llamado *producto agudo*, como sigue.

Sean PI_i y PI_j implicantes primos de una función, cada uno de los cuales representa el conjunto de mintérminos cubiertos por ese implicante primo. El *producto agudo*, $PI_i \# PI_j$, se define como el conjunto de mintérminos cubiertos por PI_i no cubiertos por PI_j; es decir, el conjunto PI_i sin los mintérminos comunes a PI_i y PI_j. En términos de operaciones de conjuntos,

$$PI_i \# PI_j = PI_i - (PI_i \cap PI_j). \tag{3.1}$$

Dado un conjunto de implicantes primos PI = $\{P_1,...,P_n\}$ que cubre una función, el producto agudo

$$S = P_1 \#\{P_2, \ldots, P_n\} = (\ldots((P_1 \# P_2) \# P_3) \ldots \# P_n)$$

toma el conjunto de mintérminos cubiertos por P_1 y elimina de éste todos los mintérminos cubiertos por al menos algún otro implicante primo de PI. Si S no está vacío, contiene los mintérminos cubiertos por P_1 y por ningún otro implicante primo de $\{P_2,...,P_n\}$. Así, P_1 es esencial para la función y debe quedar incluido en EPI. Si S está vacío, cada mintérmino cubierto por P_1 también queda cubierto por al menos otro implicante primo de $\{P_2,...,P_n\}$ lo que hace a P_1 no esencial para la función.

Para las funciones con especificación incompleta, donde DC es el conjunto de términos *prescindibles*, calculamos el producto agudo $S\#DC$ eliminando los términos prescindibles de S. Si el resultado es un conjunto vacío, los únicos términos cubiertos exclusivamente por P_1 son términos prescindibles y, por tanto, P_1 no es esencial para la función. Esta verificación garantiza que sólo consideremos términos especificados (no prescindibles) para la cubierta de la función.

Implicantes primos esenciales mediante consenso iterativo

Podemos extender los algoritmos de consenso iterativo y generalizado para identificar los implicantes primos esenciales de una función a partir de su conjunto de implicantes primos. Dado el conjunto de implicantes primos PI = $\{P_1, P_2,...P_n\}$, podemos determinar si cada implicante primo P_i es esencial o no de la manera siguiente.

Eliminamos P_i de PI y aplicamos uno de los algoritmos de consenso al conjunto restante de implicantes primos (PI – P_i). Los dos algoritmos de consenso garantizan la obtención del conjunto completo de implicantes primos para cualquier conjunto de implicantes que describa a una función. Por tanto, si regeneramos P_i, entonces (PI – P_i) cubre la función; es decir, cada mintérmino cubierto por Pi también queda cubierto por al menos otro implicante primo de (PI – P_i). Por tanto, P_i no es esencial. Si no se regenera P_i, éste cubre al menos un mintérmino no cubierto por alguno de los otros implicantes primos de PI, de modo que es esencial para la función y debe estar incluido en EPI.

3.11.4 Cómo completar una cubierta mínima

Si los implicantes primos esenciales cubren todos los mintérminos de una función, entonces el conjunto EPI de implicantes primos esenciales es una cubierta

mínima única de la función. En caso contrario, hay que elegir algún o algunos otros implicantes primos para completar la cubierta.

Los algoritmos de minimización utilizan una gran variedad de métodos para extraer una cubierta mínima de un conjunto de implicantes primos después de eliminar los implicantes primos esenciales. Se aplican diversas *heurísticas*, basadas en las propiedades de la función o de la estructura de circuito deseada, para guiar el proceso de solución. El objetivo de un algoritmo de minimización es determinar la solución de costo mínimo, es decir, el conjunto de implicantes primos no esenciales más costeable que complete una cubierta. Sin embargo, dado el costo de determinación de una solución óptima, en particular para las funciones con varias salidas, con frecuencia debemos establecer ciertas concesiones, buscando una solución cercana a la óptima pero que se pueda generar en un lapso razonable de cómputo.

La figura 3.38 muestra el algoritmo básico para determinar la cubierta mínima de una función, dado el conjunto de implicantes primos esenciales EPI y el conjunto restante de implicantes primos no esenciales, PI. A continuación explicaremos cada paso del proceso.

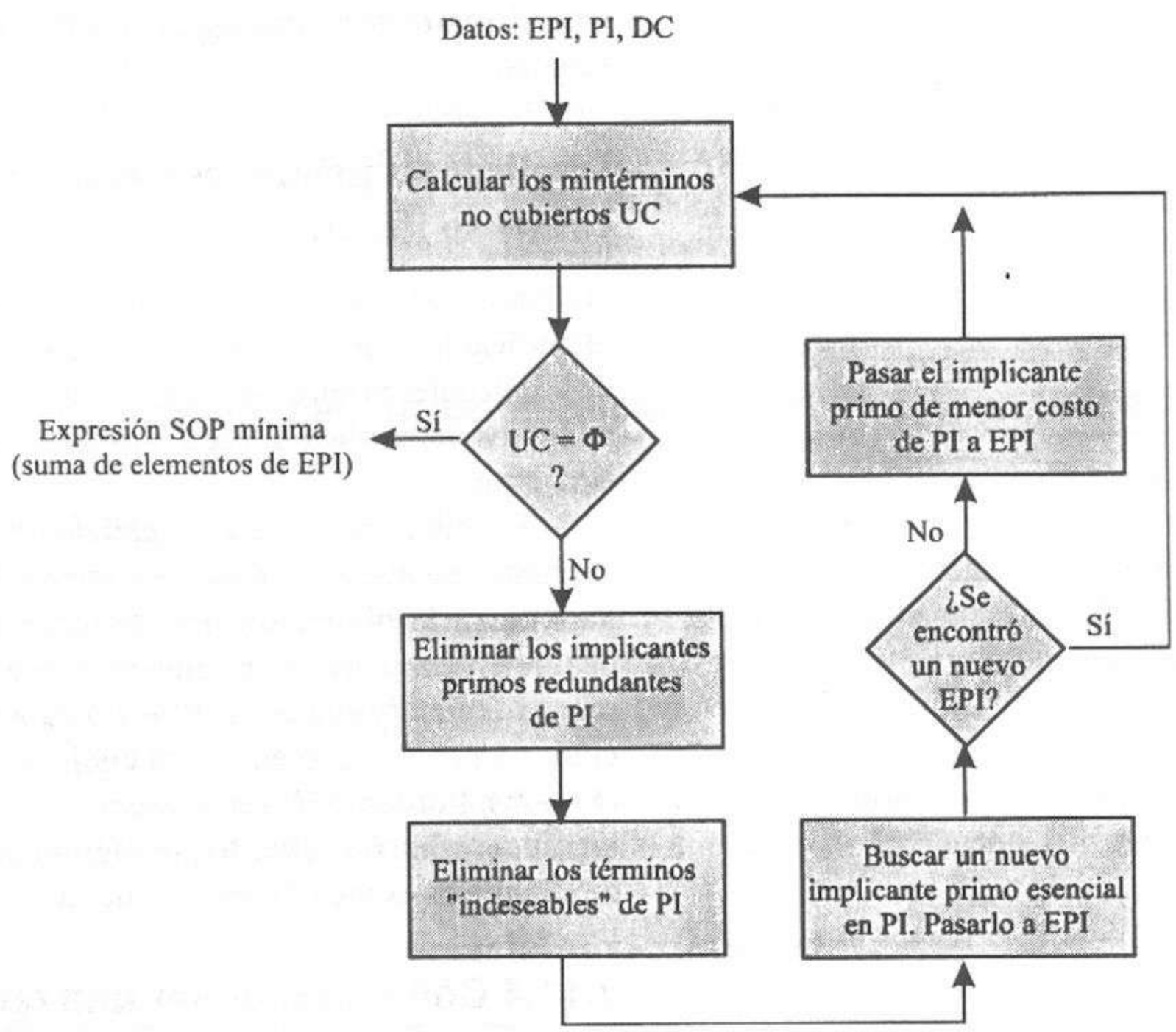


Figura 3.38 Selección de la cubierta mínima de una función.

Determinación de los mintérminos restantes no cubiertos

Después de calcular EPI, debemos determinar los mintérminos de la función no cubiertos por EPI, si existen. Con este fin, podemos utilizar el producto agudo de la manera siguiente para eliminar de PI los mintérminos cubiertos por EPI y los términos prescindibles.

$$\text{UC} = (\text{PI}\#\text{EPI})\#\text{DC} \tag{3.2}$$

El conjunto resultante, UC, contiene todos los mintérminos no cubiertos de la función. Si UC = Φ, no faltan mintérminos por cubrir y EPI cubre por completo a la función. En caso contrario, hay que elegir uno o más implicantes primos para cubrir los elementos de UC.

Eliminación de los implicantes primos redundantes

Para simplificar la selección de términos de PI que cubran los términos restantes de UC, primero eliminamos de PI los implicantes primos *redundantes*; es decir, los que no cubren cubos de UC. Si determinamos que un implicante primo es redundante, éste no puede contribuir a la solución y debemos dejar de tomarlo en cuenta eliminándolo de PI. Podemos identificar un implicante primo P_i como redundante si

$$\text{UC} \cap P_i = \Phi \tag{3.3}$$

Podemos calcular la intersección UC $\cap$ P_i mediante el producto STAR ya descrito.

Eliminación de implicantes primos indeseables

Después de eliminar los implicantes primos redundantes de PI, debemos determinar si alguno de los implicantes primos restantes tiene un mejor costo que los demás para completar la cubierta. Un implicante primo P_i es *indeseable* como opción para la solución si existe algún otro implicante primo P_j tal que P_j cubre los mismos mintérminos no cubiertos y tiene un menor costo (tiene menos literales). En otras palabras, la selección de P_i no puede conducir a una solución de menor costo que P_j y, por tanto, hay que eliminarlo de PI. P_i es indeseable si en el conjunto PI existe un implicante primo P_j tal que

$$(\text{UC} \cap P_i)\#P_j = \Phi \tag{3.4}$$

y

$$\text{costo } Pi \geq \text{costo } Pj \tag{3.5}$$

La ecuación 3.4 indica que P_j cubre todos los mintérminos de UC cubiertos por P_i, y la ecuación 3.5 indica que P_j es de menor o igual costo.

Selección de implicantes primos no esenciales

Después de eliminar los implicantes primos redundantes o indeseables de PI, debemos determinar si la eliminación de estos términos hace que alguno de los cubos restantes en PI sea esencial. Esto ocurre si hemos eliminado todos excepto uno de los implicantes primos que cubren un elemento de UC. Podemos identificar los implicantes primos vueltos que se hacen esenciales de esta forma aplicando de nuevo el algoritmo ya descrito para determinar los implicantes primos esenciales de una función.

Si ninguno de los implicantes primos de PI resulta ser esencial, hay un *ciclo*. En este caso, descomponemos el ciclo eligiendo un término arbitrario de PI para agregarlo a EPI. El proceso de reducción del conjunto PI y la selección de otros implicantes primos para agregarlos a EPI continúa hasta que UC = Φ. Sin embargo, siempre que hacemos una elección arbitraria, es posible que la solución final no sea óptima. Muchos programas de CAD están diseñados de modo que calculan todas las posibles soluciones y evalúan los resultados de cada posible decisión a fin de determinar la mejor opción para interrumpir un ciclo. Sin embargo, en el caso de funciones de gran tamaño, esto podría requerir un tiempo de cómputo mayor que lo práctico, de modo que basta aceptar los resultados obtenidos al realizar una elección para romper el ciclo.

3.11.5 Otros algoritmos de minimización

En esta sección sólo analizamos las funciones con una salida. Sin embargo, la mayor parte de los algoritmos algebraicos presentados se ha extendido a la minimización simultánea de varias funciones; esta extensión es necesaria para los circuitos lógicos con varias salidas, lo cual incluye muchas aplicaciones prácticas. El lector debe consultar [9–12] si desea más información acerca de la extensión de los algoritmos de consenso iterativo y generalizado al caso de las funciones con varias salidas.

Los algoritmos de minimización difieren principalmente en los métodos utilizados para extraer una cubierta mínima de una función a partir de un conjunto de implicantes primos. La heurística que usan muchos de estos algoritmos se enfoca a estructuras de circuitos específicas, aprovechando las características particulares de estas estructuras como una guía en la búsqueda de una solución óptima. Un ejemplo básico es el arreglo lógico programable (PLA) que describiremos en el capítulo 5. Básicamente, un PLA realiza expresiones de suma de productos formando términos producto mediante un arreglo AND programable y formando a continuación sumas de estos términos producto con un arreglo OR programable. En general, los circuitos de PLA se caracterizan por tener un número relativamente grande de entradas y un número relativamente pequeño de términos producto. En consecuencia, los algoritmos que requieren la generación y manipulación de todos los mintérminos de la función son bastante menos eficaces que los que trabajan en forma directa con los términos producto.

Uno de los programas para minimización de PLAs de uso más amplio es ESPRESSO-II, desarrollado en la Universidad de California en Berkeley [13]. ESPRESSO-II se creó después de un amplio estudio de dos algoritmos de minimización existentes: MINI, desarrollado en IBM en 1974 [14] y PRESTO [15]. En esencia, ESPRESSO-II adoptó las mejores características de sus predecesores, proporcionando algoritmos más eficientes para los pasos individuales y

trabajando después con las etapas finales del algoritmo para completar de manera más eficiente la cubierta mínima una vez determinados los implicantes primos esenciales. Desde que se creó ESPRESSO-II, se han desarrollado muchos otros programas para minimización. El lector debe consultar la documentación anexa a las herramientas de CAD disponibles en su lugar de trabajo para identificar los algoritmos disponibles con estas herramientas.

3.12 Resumen

Hemos presentado métodos gráficos y tabulares para minimizar las funciones de conmutación. Analizamos con detalle las ramificaciones de cada técnica. La técnica gráfica utiliza el mapa de Karnaugh que, como mostramos, sólo es una representación conveniente del diagrama de Venn. El método de Quine-McCluskey, que es un método tabular, utiliza una búsqueda lineal eficiente en el proceso de minimización. Esta técnica de minimización también es adecuada para su programación en una computadora digital. Presentamos el algoritmo de Petrick para la deducción de las cubiertas mínimas de una función a partir de su implicante primo, como ejemplo de método óptimo. Por último, analizamos las técnicas de minimización que se pueden programar de manera eficiente para su inclusión en los sistemas asistidos por computadora, incluidos los algoritmos de consenso iterativo y generalizado.

REFERENCIAS BIBLIOGRÁFICAS

1. M. KARNAUGH, "The Map Method of Combinational Logic Circuits," *AIEE Comm. Electronics*, November 1953, pp. 593–599.
2. E. W. VEITCH, "A Chart Method for Simplifying Truth Functions," *Proc. Computing Machinery Conf.*, May 2–3, 1952, pp. 127-133.
3. D. A. HUFFMAN, "The Design and Use of Hazard-free Switching Networks," *J. ACM*, Vol. 4, No. 1, January 1957, pp. 47–62.
4. Z. KOHAVI, *Switching and Finite Automata Theory*, 2nd ed., New York: McGraw-Hill Book Co., 1978.
5. E. J. MCCLUSKEY JR., *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965.
6. E. J. MCCLUSKEY JR., "Minimization of Boolean Functions," *Bell System Tech. J.*, November 1956, pp. 1417–1444.
7. W. V. QUINE, "The Problem of Simplifying Truth Functions," *Amer. Math. Monthly*, Vol. 59, 1952, pp. 521-531.
8. S. R. PETRICK, "On the Minimization of Boolean Functions," *Prof. Symp. Switching Theory*, ICIP, Paris, France, June 1959.
9. T. DOWNS, AND M. F. SCHULZ, *Logic Design with Pascal, Computer Aided Design Techniques*, New York: Van Nostrand Reinhold, 1988.
10. J. F. WAKERLY, *Digital Design Principles and Practices*, 2nd ed., Chapter 2. Englewood Cliffs, NJ: Prentice Hall, 1994.
11. T. H. MOTT, JR., "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants", *IRE Trans. Electronic Computers*, Vol. EC-9, No. 2, 1960, pp. 245–252.

12. P. TISON, "Generalization of Consensus Theory and Application to the Minimization of Boolean Functions", *IEEE Trans. Electronic Computers*, Vol. EC-16, No. 4, 1967, pp. 446–456.

13. R. K. BRAYTON, G.C. HACHTEL, C.T. MCMULLEN, AND A.L. SANGIOVANNI-VINCENTELLI, *Logic Minimization Algorithms for VLSI Synthesis*, Boston: Kluwer Academic Publishers, 1984.

14. S. J. HONG, R.G. CAIN, AND D.L. OSTAPKO, "MINI: A Heuristic Approach for Logic Minimization", *IBM J. of Res. and Dev.*, Vol. 18, pp 443–458, September 1974.

15. D. W. BROWN, "A State Machine Synthesizer - SMS", *Proc. 18th Design Automation Conference*, pp 301–304, Nashville, June 1981.

PROBLEMAS

3.1 Grafique las siguientes funciones sobre el mapa de Karnaugh.

(a) $f(A, B, C) = \bar{A}\bar{B} + \bar{B}C + \bar{A}C$

(b) $f(A, B, C, D) = \bar{B}\bar{C}D + \bar{A}B\bar{C} + AB\bar{D}$

(c) $f(A, B, C, D, E) = \bar{B}\bar{C}\bar{E} + \bar{B}CE + C\bar{D}E + \bar{A}BC\bar{D} + AB\bar{C}D\bar{E}$

3.2 **Minimice las siguientes funciones mediante el mapa K.**

(a) $f(A, B, C) = \sum m(3, 5, 6, 7)$

(b) $f(A, B, C, D) = \sum m(0, 1, 4, 6, 9, 13, 14, 15)$

(c) $f(A, B, C, D) = \sum m(0, 1, 2, 8, 9, 10, 11, 12, 13, 14, 15)$

(d) $f(A, B, C, D, E) = \sum m(3, 4, 6, 9, 11, 13, 15, 18, 25, 26, 27, 29, 31)$

(e) $f(A, B, C, D, E) = \sum m(1, 5, 8, 10, 12, 13, 14, 15, 17, 21, 24, 26, 31)$

3.3 **Minimice las siguientes funciones con términos prescindibles usando el mapa K.**

(a) $f(A, B, C, D) = \sum m(2, 9, 10, 12, 13) + d(1, 5, 14)$

(b) $f(A, B, C, D) = \sum m(1, 3, 6, 7) + d(4, 9, 11)$

(c) $f(A, B, C, D, E) = \sum m(3, 11, 12, 19, 23, 29) + d(5, 7, 13, 27, 28)$

3.4 El circuito de la figura P3.1 acepta entradas BCD para los dígitos digitales 0 a 9. La salida debe ser 1 sólo si la entrada es impar. Diseñe el circuito lógico mínimo para llevar a cabo esto.

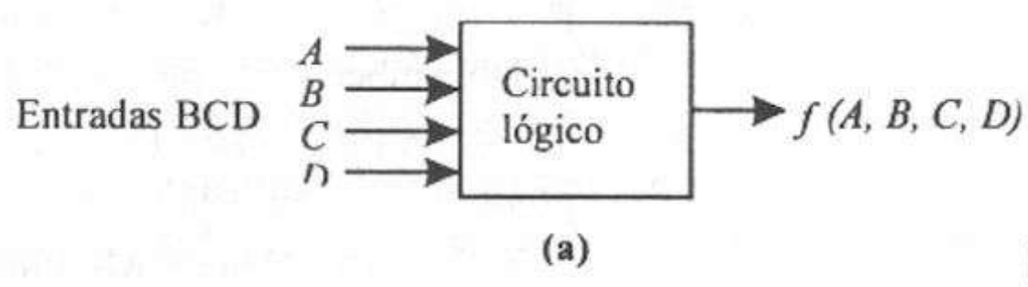


Figura P3.1

3.5 Utilice el mapa K para desarrollar las siguientes funciones POS a su forma canónica.

(a) $f(A, B, C) = (A + \bar{B})(\bar{A} + B)(B + \bar{C})$

(b) $f(A, B, C, D) = (A + \bar{D})(\bar{A} + C)$

3.6 **Minimice las siguientes funciones mediante un mapa K.**

(a) $f(A, B, C, D) = \sum m(3, 4, 6, 8, 9, 12, 14)$

(b) $f(A, B, C, D, E)$

$$= \sum m(1, 3, 4, 9, 11, 12, 13, 15, 17, 19, 22, 25, 27, 29, 30, 31)$$

3.7 Utilice mapas K para desarrollar las siguientes funciones de conmutación a su forma SOP canónica.

(a) $f(A, B, C) = (\bar{A} + B)(A + B + \bar{C})(\bar{A} + C)$

(b) $f(A, B, C, D) = A\bar{B} + \bar{A}CD + B\bar{C}\bar{D}$

(c) $f(A, B, C, D) = (A + \bar{B})(C + \bar{D})(\bar{A} + C)$

(d) $f(A, B, C, D, E) = \bar{A}E + BCD$

3.8 Determine cuáles de las siguientes funciones son equivalentes.

$$f_1(A, B, C, D) = AC + BD + A\bar{B}\bar{D}$$
$$f_2(A, B, C, D) = A\bar{B}\bar{D} + AB + \bar{A}B\bar{C}$$
$$f_3(A, B, C, D) = BD + A\bar{B}\bar{D} + ACD + ABC$$
$$f_4(A, B, C, D) = AC + A\bar{B}\bar{C}\bar{D} + \bar{A}BD + B\bar{C}D$$
$$f_5(A, B, C, D) = (B + \bar{D})(A + B)(A + \bar{C})$$

3.9 Utilice un mapa K para determinar las siguientes formas de la función de conmutación dada.

(a) Forma SOP canónica **(b)** Forma POS canónica

$$f(A, B, C, D, E) = B\bar{D}E + A\bar{B}D + \bar{A}C\bar{D}E + A\bar{C}E$$

3.10 Diseñe una red combinatoria con varias salidas que tenga dos señales de entrada x_0 y x_1, dos señales de control c_0 y c_1, y dos funciones de salida, f_0 y f_1. Las señales de control tienen el siguiente efecto sobre las salidas:

c_0	c_1	f_0	f_1
0	0	0	0
0	1	x_0	0
1	0	0	x_1
1	1	x_0	x_1

Por ejemplo, cuando $c_0 = 0$ y $c_1 = 1$, entonces $f_0(x_0,x_1,c_0,c_1) = x_0$ y $f_1(x_0,x_1,c_0,c_1) = 0$.

3.11 Utilice mapas K para determinar las siguientes funciones.

$$f_1(A, B, C, D) = f_\alpha(A, B, C, D) \cdot f_\beta(A, B, C, D)$$
$$f_2(A, B, C, D) = f_\alpha(A, B, C, D) + f_\beta(A, B, C, D)$$
$$f_3(A, B, C, D) = \bar{f}_1(A, B, C, D) \cdot f_2(A, B, C, D)$$

donde $$f_4(A, B, C, D) = f_\alpha(A, B, C, D) \oplus f_\beta(A, B, C, D)$$

$$f_\alpha(A, B, C, D) = AB + BD + \bar{A}\bar{B}C$$
$$f_\beta(A, B, C, D) = \bar{A}B + B\bar{D}$$

3.12 **(a)** Utilice mapas K para generar todos los implicantes primos de la siguiente red lógica de salida.

$$f_\alpha(A, B, C, D) = AB + BD + \bar{A}\bar{B}C$$
$$f_\beta(A, B, C, D) = \bar{A}B + B\bar{D}$$

(b) Repita la parte (a) con la técnica de Quine-McCluskey. Compare las tablas de implicantes primos.

3.13 Minimice las siguientes funciones empleando el método de Quine-McCluskey.

(a) $f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 11, 12)$

(b) $f(A, B, C, D, E) = \sum m(0, 1, 2, 7, 9, 11, 12, 23, 27, 28)$

3.14 Utilice el método de Quine-McCluskey para minimizar las siguientes funciones con términos prescindibles.

(a) $f(A, B, C, D) = \sum m(0, 6, 9, 10, 13) + d(1, 3, 8)$

(b) $f(A, B, C, D) = \sum m(1, 4, 7, 10, 13) + d(5, 14, 15)$

3.15 Minimice las siguientes funciones con varias salidas mediante la técnica de Quine-McCluskey.

(a) $f_\alpha(A, B, C, D) = \sum m(0, 1, 2, 9, 15)$
$f_\beta(A, B, C, D) = \sum m(0, 2, 8, 12, 15)$

(b) $f_\alpha(A, B, C, D) = \sum m(3, 7, 9, 14) + d(1, 4, 6, 11)$
$f_\beta(A, B, C, D) = \sum m(6, 7, 12) + d(3, 14)$

3.16 Diseñe una red de conmutación que acepte entradas BCD y proporcione una salida 1 sólo cuando el dígito decimal de entrada sea divisible entre 3. Utilice un mapa K de 4 variables para diseñar su circuito.

3.17 Diseñe una red de conmutación que tenga cinco variables de entrada y una variable de salida. Cuatro de las variables de entrada representan dígitos BCD, y la quinta es una línea de control. Si la línea de control asume el valor lógico 0, la salida debe ser 1 sólo si el dígito BCD es mayor o igual que 5. Si la línea de control está alta, la salida debe ser 1 sólo si el dígito BCD es menor o igual que 5.

3.18 Diseñe una red lógica con varias salidas, cuya entrada sea un dígito BCD y cuyas salidas se definen como sigue: f_1 detecta los dígitos de entrada divisibles entre 4; f_2 detecta los números mayores o iguales que 3; f_3 detecta los números menores que 7.

3.19 Aplique el procedimiento de cubierta para obtener una lista mínima de implicantes primos para la función

$$f(A, B, C, D) = \sum m(1, 3, 4, 6, 7, 9, 13, 15)$$

3.20 Indique las siguientes funciones sobre el mapa K y determine las listas de mintérminos.

(a) $f(A, B, C) = \bar{B} + A\bar{C}$

(b) $f(A, B, C) = \bar{A}\bar{C} + \bar{A}B + BC$

3.21 Indique las siguientes funciones sobre el mapa K y determine las listas de mintérminos.

(a) $f(A, B, C) = \bar{A}B + BC + AC + A\bar{B}$

(b) $f(A, B, C) = \bar{B}C + \bar{A}B + B\bar{C}$

3.22 Indique las siguientes funciones sobre el mapa K y determine las listas de mintérminos.

(a) $f(A, B, C, D) = \bar{A}\bar{B}C + A\bar{C}\bar{D} + B\bar{C}D + AB\bar{D}$

(b) $f(A, B, C, D) = \bar{A}\bar{B}\bar{C} + \bar{B}CD + AB\bar{D} + ABC$

3.23 Indique las siguientes funciones sobre el mapa K y determine las listas de mintérminos.

(a) $f(A, B, C, D) = \bar{B}CD + \bar{A}B\bar{D} + B\bar{C}D + A\bar{B}D$

(b) $f(A, B, C, D) = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{C}D + BCD + ABC$

3.24 Indique las siguientes funciones sobre el mapa K y determine las listas de mintérminos.

(a) $f(A, B, C, D, E) = \bar{B}\bar{C}D + \bar{B}DE + \bar{A}BC\bar{D} + BC\bar{D}E + \bar{A}B\bar{D}\bar{E}$
$+ B\bar{C}\bar{D}\bar{E} + AB\bar{C}\bar{E}$

(b) $f(A, B, C, D, E) = \bar{A}\bar{B}D\bar{E} + \bar{A}B\bar{D} + BE + A\bar{B}\bar{C}\bar{D} + A\bar{C}\bar{D}E$

3.25 Indique las siguientes funciones sobre el mapa K y determine las listas de maxtérminos.

(a) $f(A, B, C) = (A + B)(\bar{B} + C)$

(b) $f(A, B, C) = \bar{B}(\bar{A} + C)$

3.26 Indique las siguientes funciones sobre el mapa K y determine las listas de maxtérminos.

(a) $f(A, B, C) = A(B + \bar{C})$

(b) $f(A, B, C) = (B + C)(A + \bar{B})$

3.27 Indique las siguientes funciones sobre el mapa K y determine las listas de maxtérminos.

(a) $f(A, B, C, D) = (\bar{C} + \bar{D})(\bar{A} + \bar{B} + D)(\bar{A} + \bar{C} + \bar{D})(\bar{A} + C + D)$
$(B + C + D)$

(b) $f(A, B, C, D) = (\bar{B} + C)(A + C + \bar{D})(A + B + \bar{D})(B + \bar{C} + \bar{D})$

3.28 Indique las siguientes funciones sobre el mapa K y determine las listas de maxtérminos.

(a) $f(A, B, C, D) = (A + \bar{D})(A + \bar{B})(\bar{B} + D)(\bar{A} + C + D)$

(b) $f(A, B, C, D) = (A + \bar{B} + C)(\bar{A} + \bar{B} + \bar{D})(\bar{A} + \bar{C} + D)(B + \bar{C} + D)$

3.29 Indique las siguientes funciones sobre el mapa K y determine las listas de maxtérminos.

$$f(A, B, C, D, E) = (B + \bar{C} + \bar{D})(A + C + D)(A + \bar{B} + D)$$
$$(\bar{A} + B + D + E)(\bar{B} + D + \bar{E})$$

3.30 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C) = \sum m(1, 5, 6, 7)$

(b) $f(A, B, C) = \sum m(0, 1, 2, 3, 4, 5)$

3.31 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C) = \sum m(0, 2, 3, 5)$

(b) $f(A, B, C) = \sum m(0, 3, 4, 6, 7)$

3.32 Simplifique las siguientes funciones mediante un mapa K.

(a) $f(A, B, C, D) = \sum m(0, 2, 5, 7, 8, 10, 13, 15)$

(b) $f(A, B, C, D) = \sum m(1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15)$

3.33 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C, D) = \sum m(0, 4, 5, 7, 8, 10, 11, 15)$

(b) $f(A, B, C, D) = \sum m(1, 4, 5, 6, 9, 11, 15)$

3.34 Simplifique las siguientes funciones mediante un mapa K.

(a) $f(A, B, C, D) = \sum m(1, 2, 5, 6, 7, 9, 11, 15)$

(b) $f(A, B, C, D) = \sum m(0, 1, 2, 5, 12, 13, 14, 15)$

3.35 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C, D) = \sum m(1, 4, 5, 6, 8, 9, 11, 13, 15)$

(b) $f(A, B, C, D) = \sum m(1, 2, 4, 5, 6, 9, 12, 14)$

3.36 Simplifique las siguientes funciones mediante un mapa K.

(a) $f(A, B, C, D, E) = \sum m(0, 4, 6, 7, 8, 11, 15, 20, 22, 24, 26, 27, 31)$

(b) $f(A, B, C, D, E) = \sum m(2, 7, 10, 12, 13, 22, 23, 26, 27, 28, 29)$

3.37 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C, D, E) = \sum m(1, 3, 8, 9, 11, 12, 14, 17, 19, 20, 22, 24, 25, 27)$

(b) $f(A, B, C, D, E) = \sum m(0, 7, 8, 10, 13, 15, 16, 24, 28, 29, 31)$

3.38 Simplifique las siguientes funciones mediante un mapa K.

(a) $f(A, B, C, D, E) = \sum m(1, 2, 5, 6, 13, 15, 16, 18, 22, 24, 29)$

(b) $f(A, B, C, D, E) = \sum m(1, 7, 9, 12, 14, 15, 16, 23, 24, 28, 30)$

3.39 Utilice el mapa K para simplificar las siguientes funciones.

(a) $f(A, B, C, D, E) = \sum m(0, 5, 10, 11, 13, 15, 16, 18, 29, 31)$
(b) $f(A, B, C, D, E) = \sum m(4, 5, 7, 8, 9, 12, 13, 16, 18, 23, 24, 25, 28, 29)$

3.40 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C) = \prod M(0, 2, 3, 4)$
(b) $f(A, B, C) = \prod M(0, 3, 4, 7)$

3.41 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C) = \prod M(0, 1, 4, 5, 6)$
(b) $f(A, B, C) = \prod M(1, 2, 3, 6)$

3.42 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C) = \prod M(1, 2, 5, 7)$
(b) $f(A, B, C) = \prod M(1, 2, 3, 4)$

3.43 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C) = \prod M(0, 1, 3, 4, 6, 7)$
(b) $f(A, B, C) = \prod M(2, 3, 5, 7)$

3.44 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \prod M(0, 1, 5, 7, 8, 10, 11, 15)$
(b) $f(A, B, C, D) = \prod M(0, 1, 2, 4, 6, 7, 8, 10, 14)$

3.45 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \prod M(2, 3, 4, 5, 7, 12, 13)$
(b) $f(A, B, C, D) = \prod M(1, 2, 5, 7, 11, 13, 15)$

3.46 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \prod M(0, 2, 4, 5, 6, 9, 11, 13)$
(b) $f(A, B, C, D) = \prod M(1, 3, 4, 5, 6, 9, 11, 12, 13)$

3.47 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \prod M(0, 1, 5, 7, 9, 11, 12, 14)$
(b) $f(A, B, C, D) = \prod M(3, 4, 5, 7, 8, 9, 10)$

3.48 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D, E) = \prod M(3, 4, 6, 13, 15, 16, 19, 24, 29, 31)$
(b) $f(A, B, C, D, E) = \prod M(1, 4, 7, 9, 15, 17, 20, 22, 25, 30)$

3.49 Determine la forma POS mínima para las siguientes funciones.
(a) $f(A, B, C, D, E) = \prod M(0, 1, 2, 5, 7, 8, 10, 15, 17, 21, 22, 24, 26, 29)$
(b) $f(A, B, C, D, E) = \prod M(0, 2, 4, 6, 9, 11, 13, 15, 16, 19, 20, 25, 27, 29, 31)$

3.50 Determine la forma SOP mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \sum m(1, 2, 7, 12, 15) + d(5, 9, 10, 11, 13)$
(b) $f(A, B, C, D) = \sum m(0, 2, 5, 15) + d(8, 9, 12, 13)$

3.51 Determine la forma SOP mínima para las siguientes funciones.
(a) $f(A, B, C, D) = \sum m(4, 7, 9, 15) + d(1, 2, 3, 6)$
(b) $f(A, B, C, D) = \sum m(0, 2, 3, 4, 5) + d(8, 9, 10, 11)$

3.52 Determine la forma SOP mínima para la siguiente función.

$$f(A, B, C, D, E) = \sum m(7, 9, 12, 13, 19, 22) + d(0, 3, 20, 25, 27, 28, 29)$$

3.53 Determine la forma POS mínima para las siguientes funciones.

(a) $f(A, B, C, D) = \prod M(4, 7, 9, 11, 12) \cdot D(0, 1, 2, 3)$

(b) $f(A, B, C, D) = \prod M(0, 3, 7, 12) \cdot D(2, 10, 11, 14)$

3.54 Determine la forma POS mínima para las siguientes funciones.

(a) $f(A, B, C, D) = \prod M(3, 4, 10, 13, 15) \cdot D(6, 7, 14)$

(b) $f(A, B, C, D) = \prod M(0, 7, 11, 13) \cdot D(1, 2, 3)$

3.55 Determine la forma POS mínima para la siguiente función.

$f(A, B, C, D, E) = \prod M(0, 5, 6, 9, 21, 28, 31) \cdot D(2, 12, 13, 14, 15, 25, 26)$

3.56 Utilice el método de Quine-McCluskey para minimizar las siguientes funciones.

(a) $f(A, B, C, D) = \sum m(0, 2, 3, 5, 7, 11, 12, 14, 15)$

(b) $f(A, B, C, D) = \sum m(0, 1, 6, 8, 9, 13, 14, 15)$

3.57 Utilice el método de Quine-McCluskey para minimizar las siguientes funciones.

(a) $f(A, B, C, D) = \sum m(1, 4, 5, 6, 8, 9, 10, 12, 14)$

(b) $f(A, B, C, D) = \sum m(4, 5, 6, 8, 11, 13, 15)$

3.58 Minimice las siguientes funciones mediante el método de Quine-McCluskey.

(a) $f(A, B, C, D) = \sum m(1, 3, 6, 7, 8, 9, 12, 14)$

(b) $f(A, B, C, D) = \sum m(0, 2, 4, 5, 10, 11, 13, 15)$

3.59 Utilice el método de Quine-McCluskey para minimizar las siguientes funciones con términos prescindibles.

(a) $f(A, B, C, D) = \sum m(1, 6, 7, 9, 12) + d(8, 11, 15)$

(b) $f(A, B, C, D) = \sum m(7, 8, 13, 15) + d(3, 4, 10, 14)$

3.60 Minimice las siguientes funciones con términos prescindibles mediante el método de Quine-McCluskey.

(a) $f(A, B, C, D) = \sum m(5, 7, 11, 12, 27, 29) + d(14, 20, 21, 22, 23)$

(b) $f(A, B, C, D) = \sum m(1, 4, 6, 9, 14, 17, 22, 27, 28) + d(12, 15, 20, 30, 31)$

3.61 Minimice las siguientes funciones con varias salidas mediante el método de Quine-McCluskey.

(a) $f_\alpha(A, B, C, D) = \sum m(4, 5, 6, 15) + d(8, 11)$
$f_\beta(A, B, C, D) = \sum m(0, 2, 3, 4, 5) + d(8, 11)$

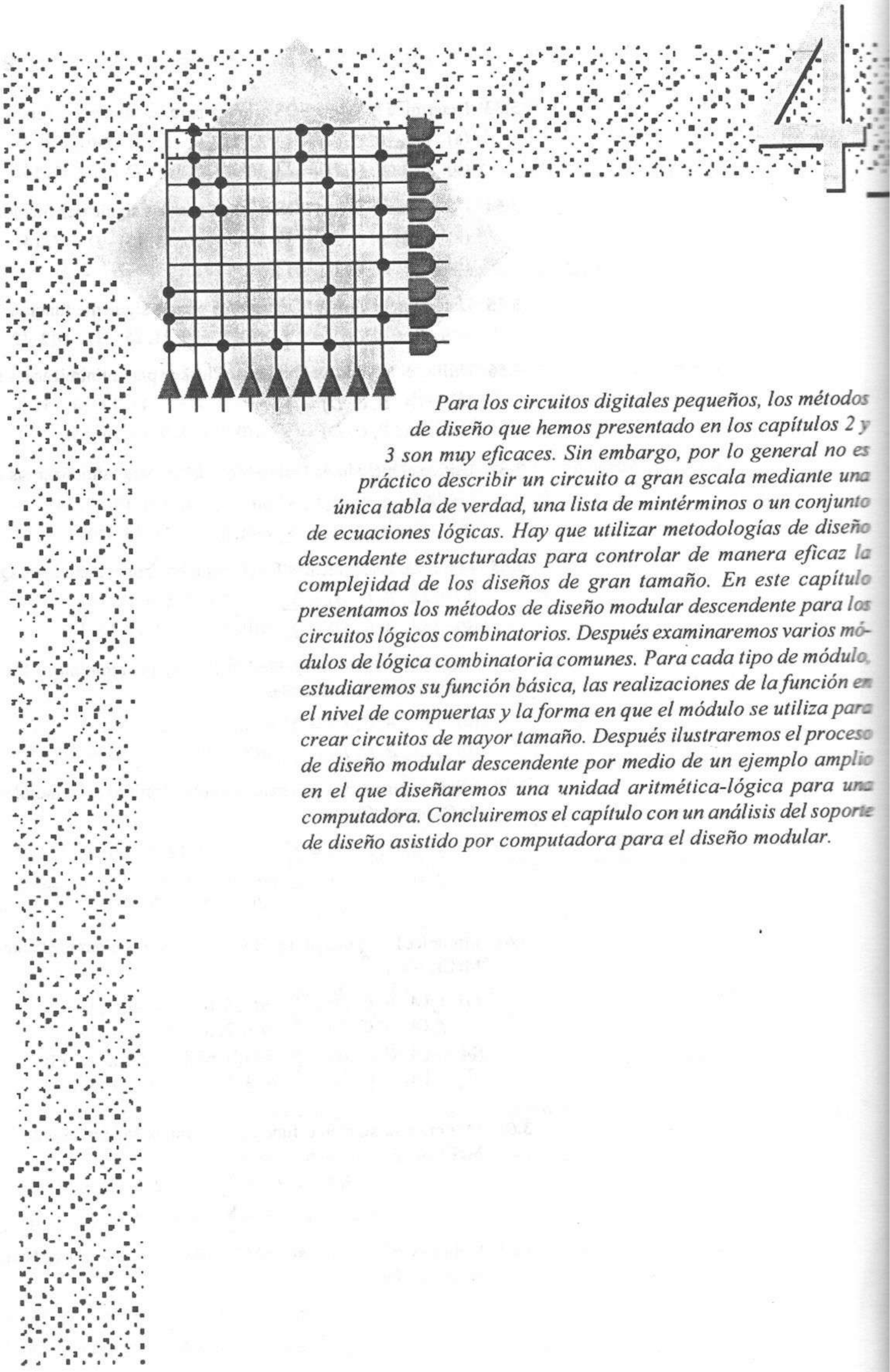
(b) $f_\alpha(A, B, C, D) = \sum m(3, 4, 6, 11, 12) + d(14, 15)$
$f_\beta(A, B, C, D) = \sum m(4, 5, 6, 11, 14) + d(8, 12)$

3.62 Minimice la siguiente función con varias salidas usando el método de Quine-McCluskey.

$$f_\alpha(A, B, C, D, E) = \sum m(0, 2, 8, 9, 20, 24) + d(4, 10, 14, 26, 30)$$
$$f_\beta(A, B, C, D, E) = \sum m(3, 4, 8, 11, 24) + d(10, 14, 20, 26, 30)$$

3.63 Utilice el método de Quine-McCluskey para minimizar la siguiente función con varias salidas.

$$f_\alpha(A, B, C, D, E) = \sum m(0, 4, 6, 20, 22) + d(2, 10, 18)$$
$$f_\beta(A, B, C, D, E) = \sum m(4, 6, 11, 19, 20, 27) + d(18, 22)$$



Para los circuitos digitales pequeños, los métodos de diseño que hemos presentado en los capítulos 2 y 3 son muy eficaces. Sin embargo, por lo general no es práctico describir un circuito a gran escala mediante una única tabla de verdad, una lista de mintérminos o un conjunto de ecuaciones lógicas. Hay que utilizar metodologías de diseño descendente estructuradas para controlar de manera eficaz la complejidad de los diseños de gran tamaño. En este capítulo presentamos los métodos de diseño modular descendente para los circuitos lógicos combinatorios. Después examinaremos varios módulos de lógica combinatoria comunes. Para cada tipo de módulo, estudiaremos su función básica, las realizaciones de la función en el nivel de compuertas y la forma en que el módulo se utiliza para crear circuitos de mayor tamaño. Después ilustraremos el proceso de diseño modular descendente por medio de un ejemplo amplio en el que diseñaremos una unidad aritmética-lógica para una computadora. Concluiremos el capítulo con un análisis del soporte de diseño asistido por computadora para el diseño modular.

Lógica combinatoria modular

4.1 Diseño modular descendente

El *diseño modular descendente* es un proceso en que inicialmente se especifica una función con un alto nivel de abstracción y después se descompone en subfunciones de menor nivel, cada una de las cuales es más concreta. El proceso de descomposición continúa hasta reducir el diseño a un conjunto de funciones, cada una de las cuales está bien definida y se puede realizar con un circuito relativamente sencillo. Por tanto, el diseño se ha desarrollado desde un nivel superior hasta un nivel inferior, en donde se pueden controlar los módulos individuales.

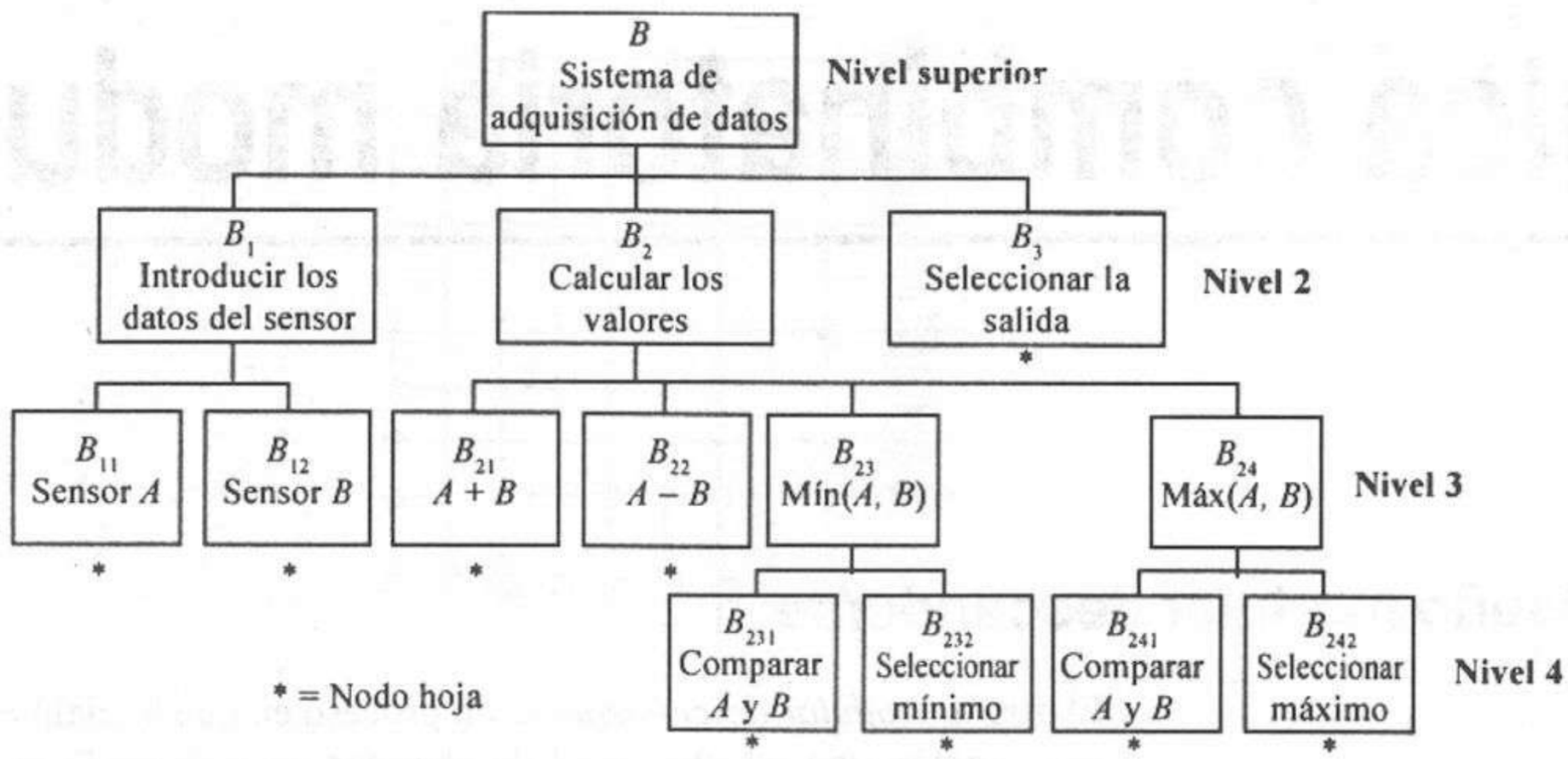
Después de definir todas las funciones, cada una de éstas se realiza mediante un módulo de circuito diseñado, implantado y probado en forma individual. Después, los módulos terminados se conectan entre sí para completar el diseño. Este proceso de implantación se conoce como proceso *ascendente*, pues comienza con los elementos inferiores del diseño y trabaja hacia la función de nivel superior.

La descomposición de un diseño se representa con frecuencia en forma de una estructura de árbol, como en la figura 4.1a. Supongamos que queremos diseñar un circuito para un sistema de adquisición de datos en el que leemos los datos de dos sensores y realizamos sobre estos valores uno de los cuatro cálculos enumerados en la tabla 4.1, seleccionándolo mediante un código de 2 bits, s_1s_2.

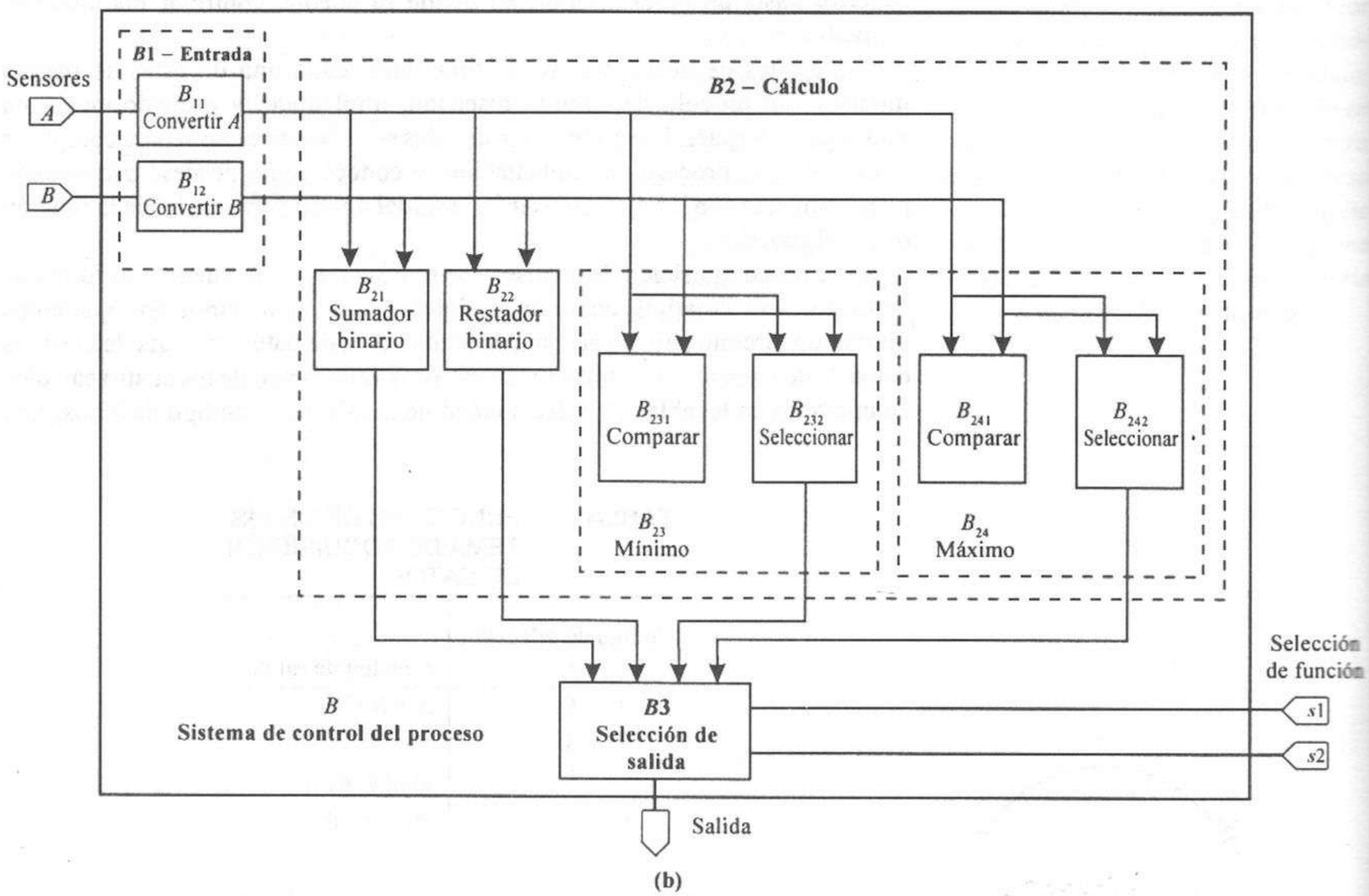
TABLA 4.1 FUNCIONES DE UN SISTEMA DE ADQUISICIÓN DE DATOS

Código de selección s_1	s_2	Función de salida
0	0	$A + B$
0	1	$A - B$
1	0	mín(A, B)
1	1	máx(A, B)

La raíz del árbol de diseño de la figura 4.1a es la función de nivel superior, B, que representa todo el sistema. Necesitamos tres funciones básicas para implantar el sistema de control del proceso: un circuito para crear señales digitales de entrada a partir de los dos sensores, un circuito para realizar los



(a)



(b)

Figura 4.1 Diseño descendente de una función. **(a)** Árbol de diseño jerárquico. **(b)** Diagrama de bloques jerárquico.

cuatro cálculos sobre los dos valores de los sensores, y un circuito para seleccionar el resultado deseado como salida del sistema, de acuerdo con el código de selección s_1s_2. Por tanto, descomponemos el bloque superior B en los bloques B_1, B_2 y B_3 del segundo nivel, que representan las funciones de entrada, cálculo y salida, respectivamente.

A continuación, examinamos cada una de las funciones de segundo nivel. La función de entrada B_1 debe convertir los datos de los dos sensores en números binarios. Por tanto, descomponemos la función B_1 en dos módulos: los circuitos de conversión B_{11} para el *sensor A* y B_{12} para el *sensor B*. Ambas funciones representan módulos que no se descomponen en circuitos de menor tamaño. Las funciones que ya no se descomponen son los *nodos hoja* del árbol de diseño.

La función de cálculo B_2 debe generar cuatro valores: $A + B$, $A - B$, Mín(A, B) y Máx(A, B). Definimos estas cuatro funciones como B_{21}, B_{22}, B_{23} y B_{24}, respectivamente. Podemos implantar las subfunciones B_{21} y B_{22} mediante circuitos sumador y restador binarios sencillos, respectivamente, por lo que son nodos hoja. Descomponemos la subfunción B_{23} en B_{231}, que compara A con B, y B_{232}, que selecciona A o B de acuerdo con el resultado de la comparación. Descomponemos la subfunción B_{24} de manera similar en B_{241} y B_{242}.

El circuito de salida es una función que selecciona uno de cuatro valores, según el código de selección s_1s_2. Podemos realizar esto con un módulo de circuito estándar; por tanto, el bloque B_3 es un nodo hoja del árbol de diseño.

Con base en el árbol de diseño de la figura 4.1a podemos desarrollar un esquema en forma de diagrama de bloques, como en la figura 4.1b. Cada bloque de este diagrama representa uno de los nodos hoja del árbol de diseño, con las entradas y salidas de todos los bloques definidas en forma precisa. Cada salida de un bloque representa la función realizada por dicho bloque, y las entradas del bloque representan los argumentos de la función.

Al terminar el diagrama de bloques, el diseñador enfrenta la decisión de si debe utilizar un módulo diseñado anteriormente o desarrollar uno nuevo para cada bloque. Se han desarrollado diversos módulos que están disponibles como funciones estándar para utilizarse como bloques de construcción de circuitos digitales complejos. En el diseño VLSI, los módulos de uso común, o *celdas estándar*, están en bibliotecas de funciones de las que pueden seleccionarse e incorporarse en los diseños conforme sea necesario. Para desarrollar tarjetas de circuitos y otros sistemas con varios circuitos, existen varios módulos estándar disponibles de manera comercial como componentes de circuitos MSI.

En las siguientes secciones examinaremos el diseño y aplicaciones de varios módulos de circuitos combinatorios estándar. Los módulos descritos en este capítulo corresponden a las funciones TTL estándar de la serie 7400, disponibles como componentes MSI en varias tecnologías, y también disponibles en la mayor parte de las bibliotecas de diseño VLSI utilizadas para crear circuitos VLSI adaptados o semiadaptados.

4.2 Decodificadores

Un *decodificador n* a 2^n es una red lógica combinatoria de varias salidas, con n líneas de entrada y 2^n señales de salida, como se muestra en la figura 4.2. Para

cada posible condición de entrada, una y sólo una señal de salida tendrá el valor lógico 1. Por tanto, podemos considerar al decodificador n a 2^n como un generador de mintérminos, donde cada salida corresponde precisamente a un mintérmino. Los decodificadores son herramientas importantes en el repertorio del diseñador. Se utilizan para cosas como interrogar a la memoria a fin de elegir una palabra específica de las que están disponibles, convertir códigos (por ejemplo, binario a decimal) y direccionar datos.

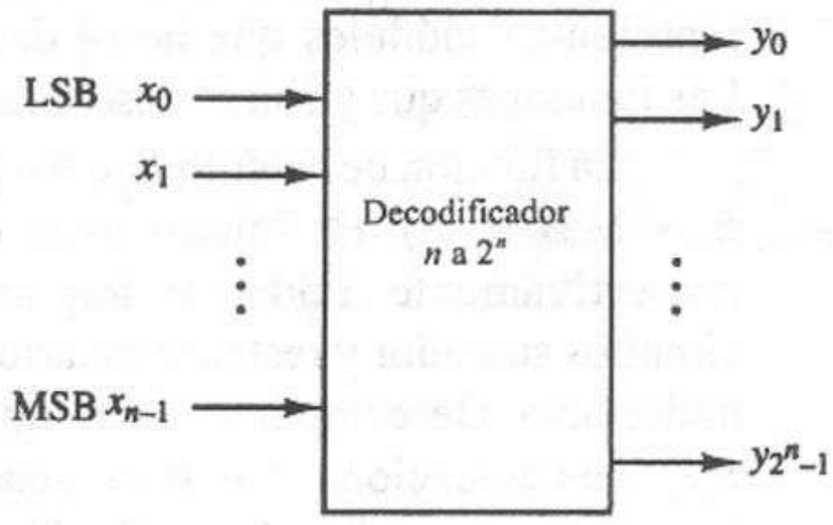


Figura 4.2 Módulo decodificador n a 2^n.

4.2.1 Estructuras de circuitos decodificadores

Antes de analizar las aplicaciones de los decodificadores n a 2^n, examinaremos algunas estructuras de circuito básicas utilizadas para implantar estos módulos.

El circuito lógico de un *decodificador paralelo* de 2 bits aparece en la figura 4.3a. En general, este decodificador es muy sencillo, aunque caro. Como podemos ver en la figura, una combinación o *vector* de entradas $BA = 00$ selecciona la línea de salida m_0, $BA = 01$ selecciona la línea de salida m_1, etcétera.

$$\begin{aligned} m_0 &= \bar{B}\bar{A} \\ m_1 &= \bar{B}A \\ m_2 &= B\bar{A} \\ m_3 &= BA \end{aligned} \tag{4.1}$$

La figura 4.3b muestra una implantación alternativa del decodificador 2 a 4 que sólo utiliza compuertas NAND. La figura 4.3c presenta otra configuración que sólo utiliza compuertas NAND sin inversores. En ambos diseños con compuertas NAND, una salida 0 indica la presencia del mintérmino correspondiente. En este caso, las salidas son *bajas activas*, pues un valor de salida 0 (el nivel "activo") significa la ocurrencia de una entrada "significativa" (la ocurrencia de un mintérmino particular). El valor de salida es 1 (el nivel "inactivo") en los demás instantes. Una señal *activa alta* utiliza el valor 1 para indicar un evento significativo y es 0 en caso contrario, como en el circuito decodificador de la figura 4.3a. También podemos considerar las señales de salida del decodificador en la figura 4.3b en forma complementada.

Observe que en las realizaciones con compuertas AND y NAND del decodificador n a 2^n que aparecen en las figuras 4.3a y b sólo existe un nivel de lógica y que se necesita una compuerta AND con n entradas para cada una de las 2^n líneas de salida. Sin embargo, pronto aparece un problema con esta configuración cuando n aumenta, pues el número de entradas a las compuertas AND (el *fan-in*) excede el límite práctico (cinco o seis). Podemos aliviar este

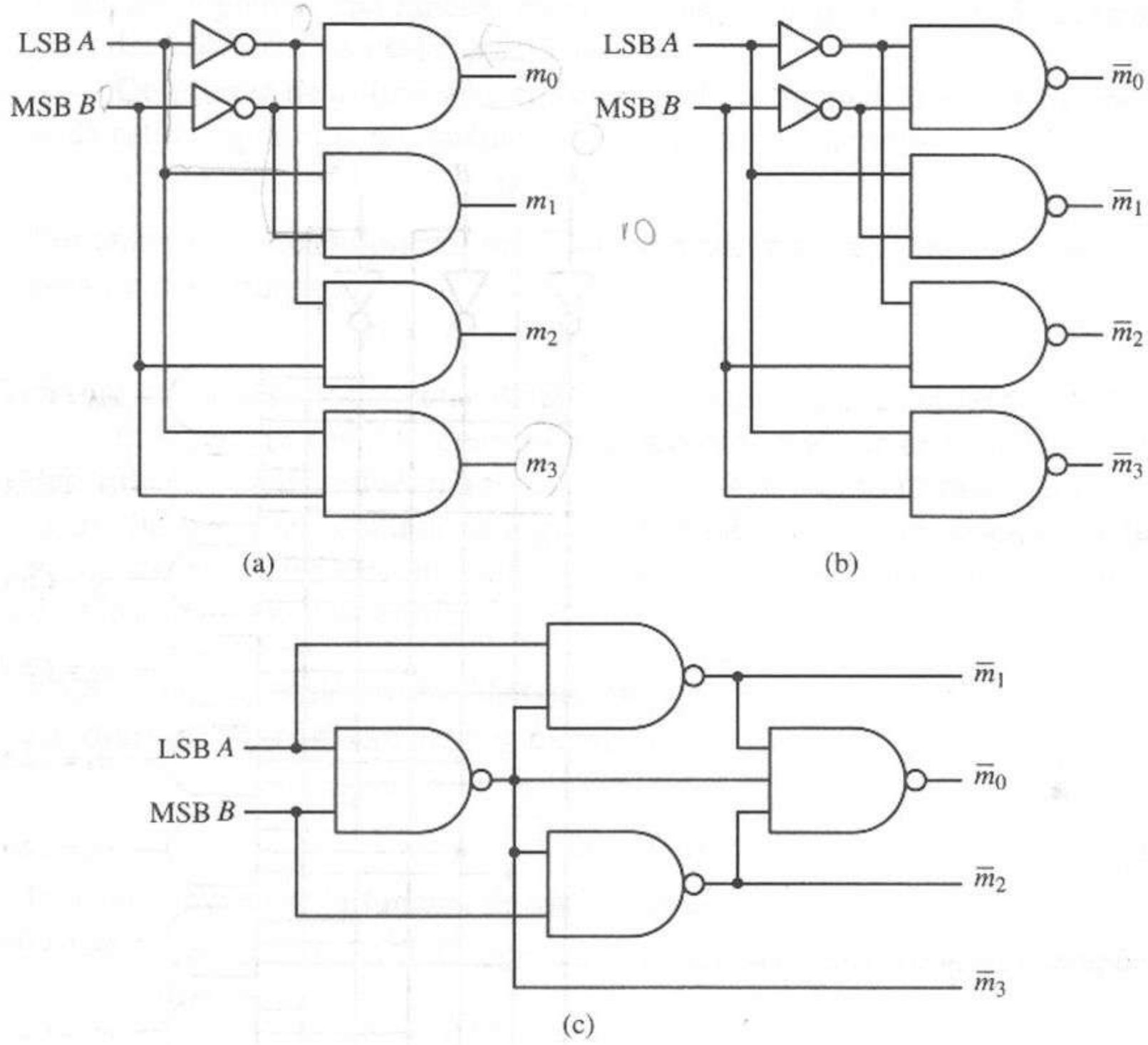


Figura 4.3 Estructuras de un circuito decodificador paralelo de dos bits. **(a)** Paralelo (salidas activas altas). **(b)** Paralelo (salidas activas bajas). **(c)** Estructura alternativa.

problema mediante un *decodificador de árbol*, como el que aparece en la figura 4.4b, que podemos comparar con el decodificador de un nivel de la figura 4.4a. Este tipo de decodificador utiliza lógica de varios niveles, exclusivamente con compuertas AND de dos entradas, sea cual sea el número de líneas de entrada. La figura 4.4c muestra una última estructura, llamada *árbol dual*, para un decodificador 4 a 16. En la estructura de árbol dual, dividimos las n líneas de entrada en grupos j y k ($j + k = n$) y después utilizamos dos decodificadores pequeños j a 2^j y k a 2^k para generar $2j$ y 2^k señales internas. A continuación usamos compuertas AND de dos entradas para combinar estas señales y formar las 2^n líneas de salida de la red total de decodificación.

4.2.2 Implantación de funciones lógicas mediante decodificadores

Las señales de salida del decodificador en forma complementada son adecuadas para su procesamiento posterior mediante la lógica NAND. Por ejemplo, si

$$f(A, B, \ldots, Z) = m_i + m_j + \cdots + m_k$$

entonces, por el teorema de DeMorgan,

$$f(A, B, \ldots, Z) = \overline{\bar{m}_i \cdot \bar{m}_j \cdot \cdots \cdot \bar{m}_k} \tag{4.2}$$

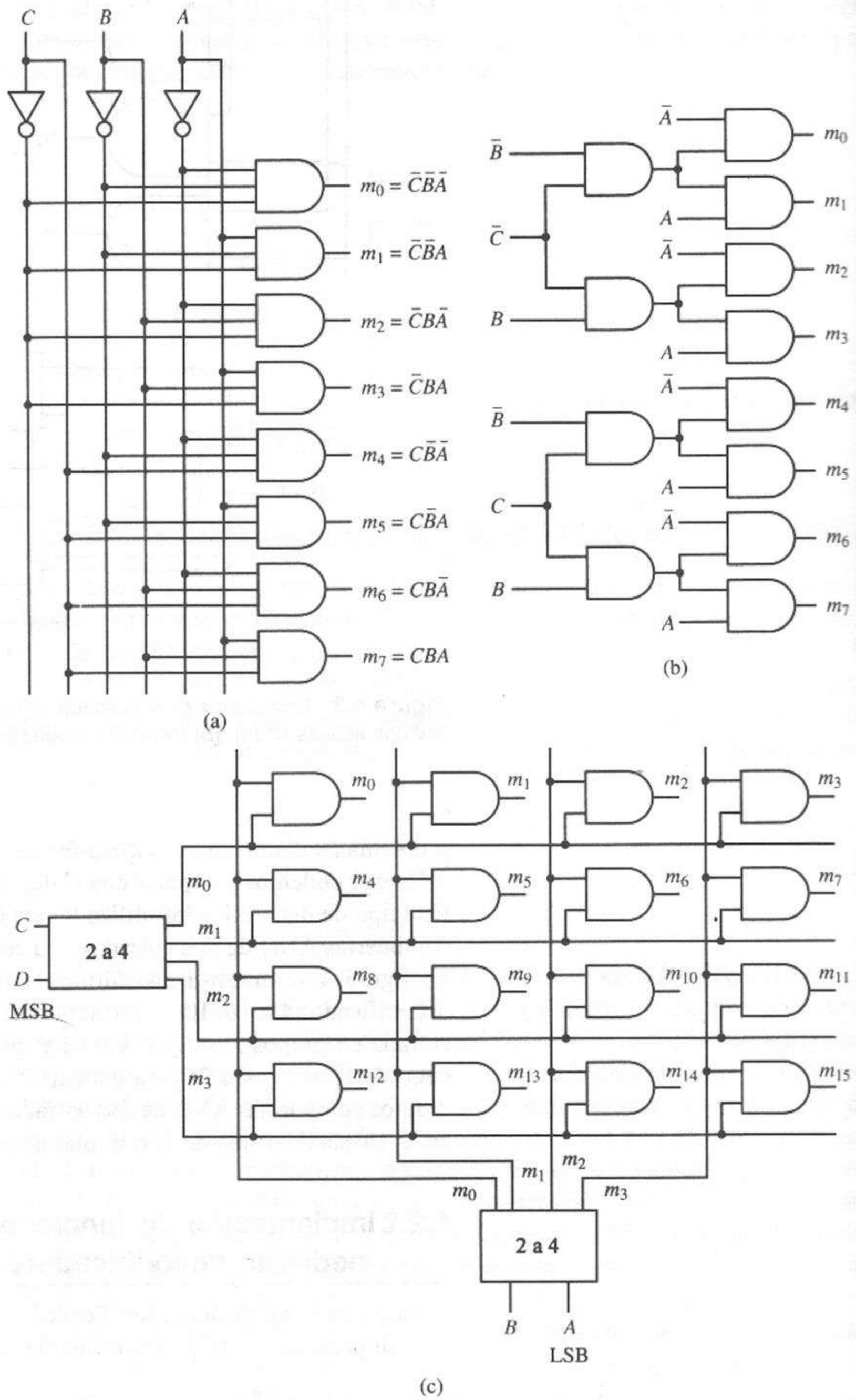


Figura 4.4 Estructuras de decodificación n a 2^n. **(a)** Decodificador de tipo paralelo de 3 bits. **(b)** Decodificador de tipo árbol de 3 bits. **(c)** Decodificador de tipo árbol dual de 4 bits.

Podemos implantar esta función mediante una compuerta NAND de k entradas y un decodificador con salidas activas bajas.

Otra forma de utilizar el decodificador de la figura 4.3b es considerar que cada salida representa un maxtérmino de una función, pues

$$M_i = \bar{m}_i$$

Por tanto, podemos implantar una función a partir de la forma canónica de su lista de maxtérminos:

$$f(A, B, \ldots, Z) = M_i \cdot M_j \cdot \cdots \cdot M_k \tag{4.3}$$

usando un decodificador con salidas activas-bajas y una compuerta AND.

El siguiente ejemplo muestra que podemos realizar una función dada a partir de su lista de mintérminos y maxtérminos de varias formas mediante un decodificador y una compuerta lógica adicional. Puesto que podemos deducir estos circuitos con facilidad, podemos examinarlos todos para determinar cuál de ellos tiene el menor costo.

EJEMPLO 4.1

Implantar las siguientes funciones lógicas mediante decodificadores y compuertas lógicas.

$$f(Q, X, P) = \sum m(0, 1, 4, 6, 7)$$
$$= \prod M(2, 3, 5)$$

Podemos implantar la función de varias formas:

1. Mediante un decodificador (con salidas activas altas) con una compuerta OR:
$$f(Q, X, P) = m_0 + m_1 + m_4 + m_6 + m_7$$

2. Usando un decodificador (con salidas activas-bajas) con una compuerta NAND:
$$f(Q, X, P) = \overline{\bar{m}_0 \cdot \bar{m}_1 \cdot \bar{m}_4 \cdot \bar{m}_6 \cdot \bar{m}_7}$$

3. Mediante un decodificador (con salidas activas altas) con una compuerta NOR:
$$f(Q, X, P) = \overline{m_2 + m_3 + m_5}$$

4. Usando un decodificador (con salidas activas-bajas) con una compuerta AND:
$$f(Q, X, P) = \bar{m}_2 \cdot \bar{m}_3 \cdot \bar{m}_5$$

Las figuras 4.5a a d muestran las cuatro implantaciones resultantes.

4.2.3 Entradas para control de activación

Los decodificadores y demás módulos funcionales incluyen con frecuencia una o más entradas de *activación*, como en la figura 4.6, que pueden servir para inhibir (desactivar) la función designada o para permitir que se realice (activar). La función de decodificación de un decodificador se inhibe haciendo que todas sus salidas pasen al estado inactivo. Por ejemplo, la salida y_0 del decodificador 2 a 4 de la figura 4.6a está dada por $y_0 = \bar{x}_1\bar{x}_0E = m_0E$. En general,

$$y_k = m_k E \tag{4.4}$$

Cuando $E = 0$, todas las salidas deben ser 0, mientras que si $E = 1$, cada salida y_k es igual a m_k.

Un uso común de la función de activación es extender la capacidad de decodificación al permitir la conexión en cascada de varios decodificadores,

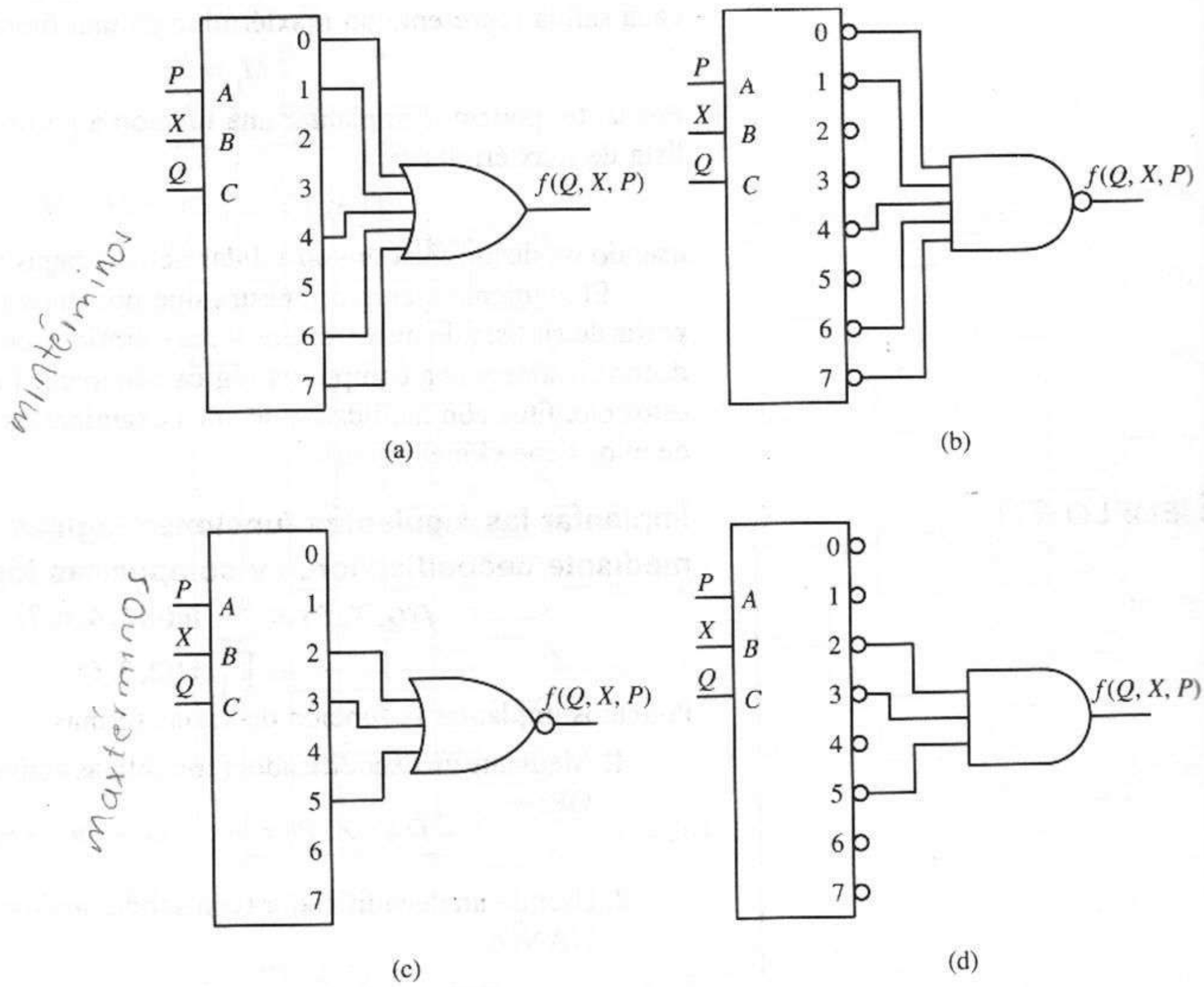


Figura 4.5 Uso de decodificadores para implantar estructuras lógicas. **(a)** Decodificador activo alto con compuerta OR. **(b)** Decodificador activo bajo con compuerta NAND. **(c)** Decodificador activo alto con compuerta NOR. **(d)** Decodificador activo bajo con compuerta AND.

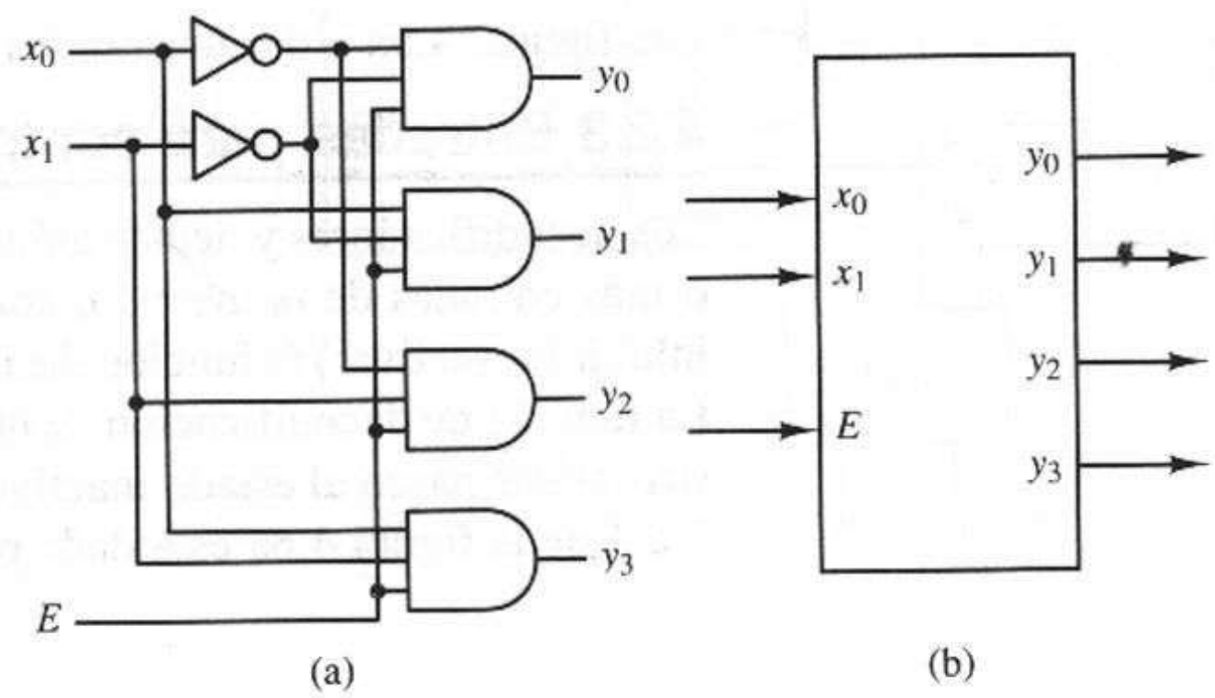


Figura 4.6 Decodificador 2 a 4 con entrada de activación *E*. **(a)** Diagrama esquemático. **(b)** Símbolo MSI.

como se muestra en la figura 4.7, la cual ilustra los decodificadores 3 a 8 y 4 a 16 realizados con el módulo decodificador 2 a 4 de la figura 4.6b. En la figura 4.7a, la entrada $I_2 = 0$ activa el decodificador superior, que así queda habilitado

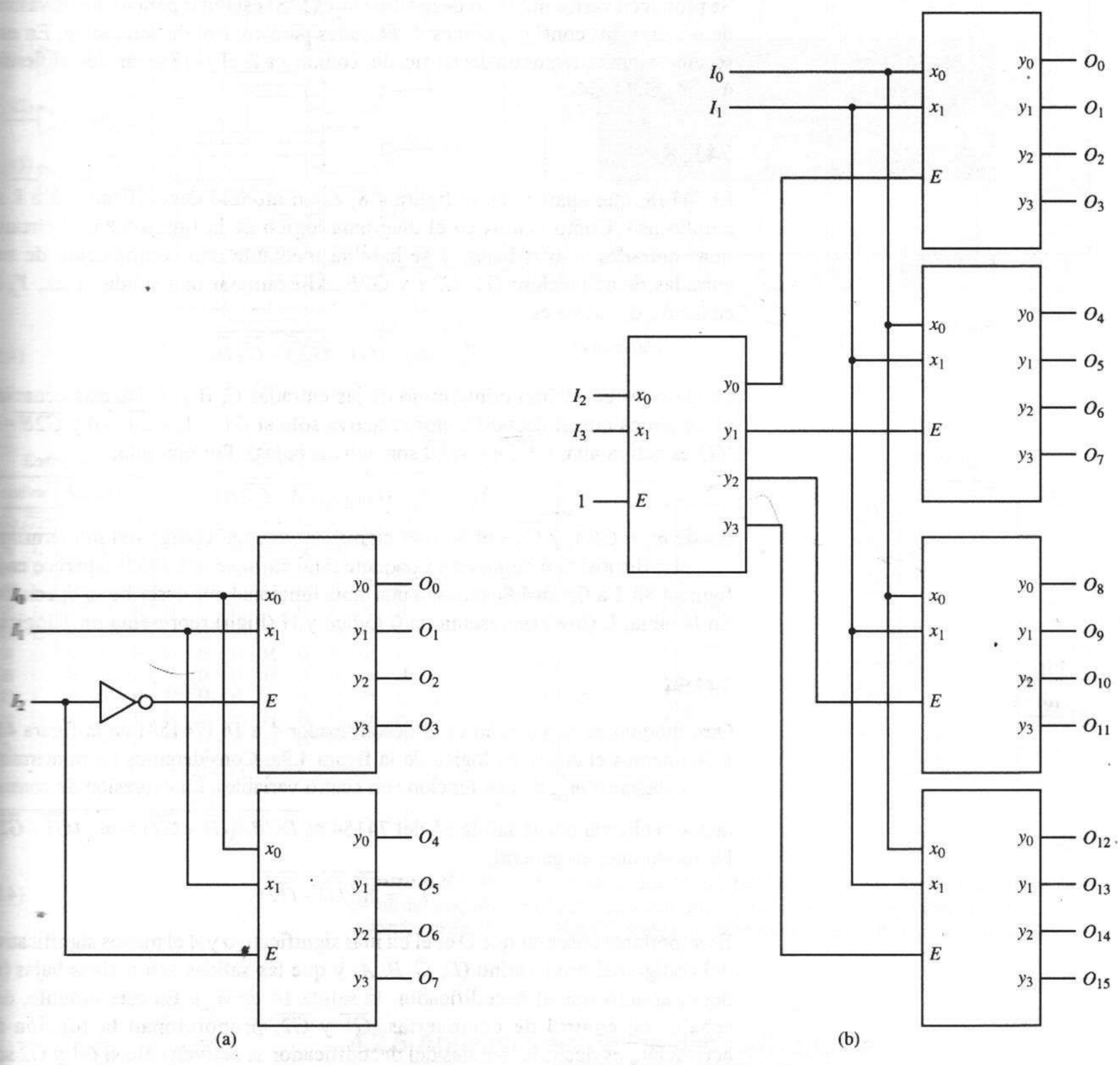
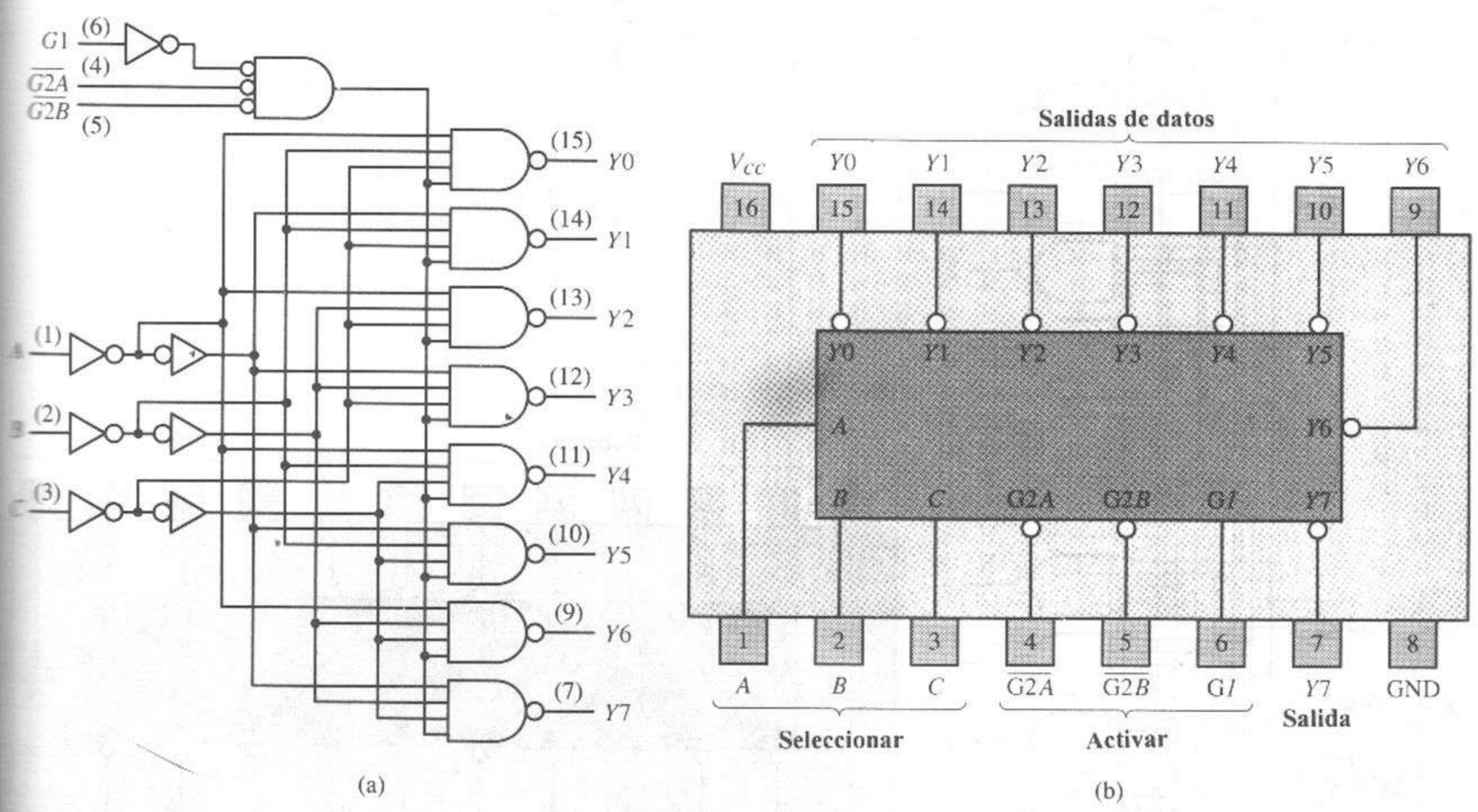


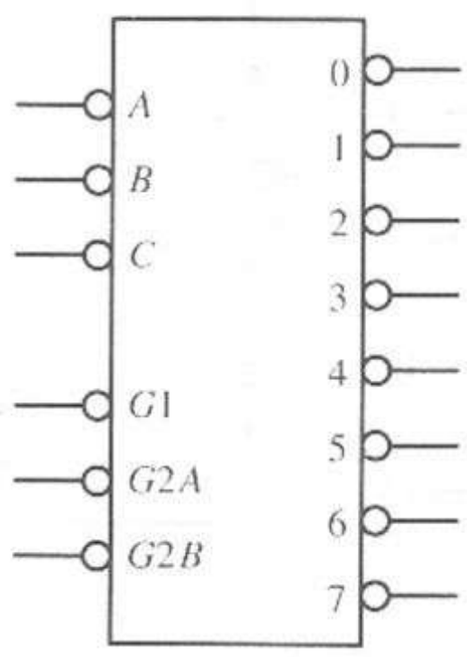
Figura 4.7 Uso de módulos decodificadores 2 a 4 para realizar decodificadores de mayor tamaño. **(a)** Decodificador 3 a 8. **(b)** Decodificador 4 a 16.



Entradas					Salidas							
Activar		Seleccionar										
$G1$	$\overline{G2}$*	C	B	A	$Y0$	$Y1$	$Y2$	$Y3$	$Y4$	$Y5$	$Y6$	$Y7$
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L
×	H	×	×	×	H	H	H	H	H	H	H	H
L	×	×	×	×	H	H	H	H	H	H	H	H

$\overline{G2}^* = \overline{G2A} + \overline{G2B}$

(c)



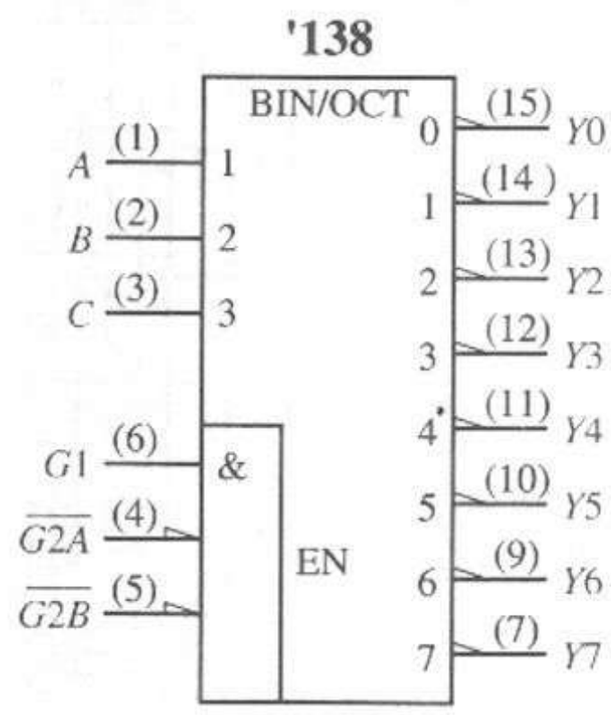


Figura 4.8 Módulo decodificador 74138. **(a)** Circuito lógico. **(b)** Configuración de *pines* del paquete. **(c)** Tabla funcional. **(d)** Símbolo genérico. **(e)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

4.2.5 Aplicaciones de los decodificadores

Decodificación de direcciones

Los decodificadores tienen muchas aplicaciones en la síntesis de las redes digitales de conmutación. Sin embargo, su uso como *decodificadores de*

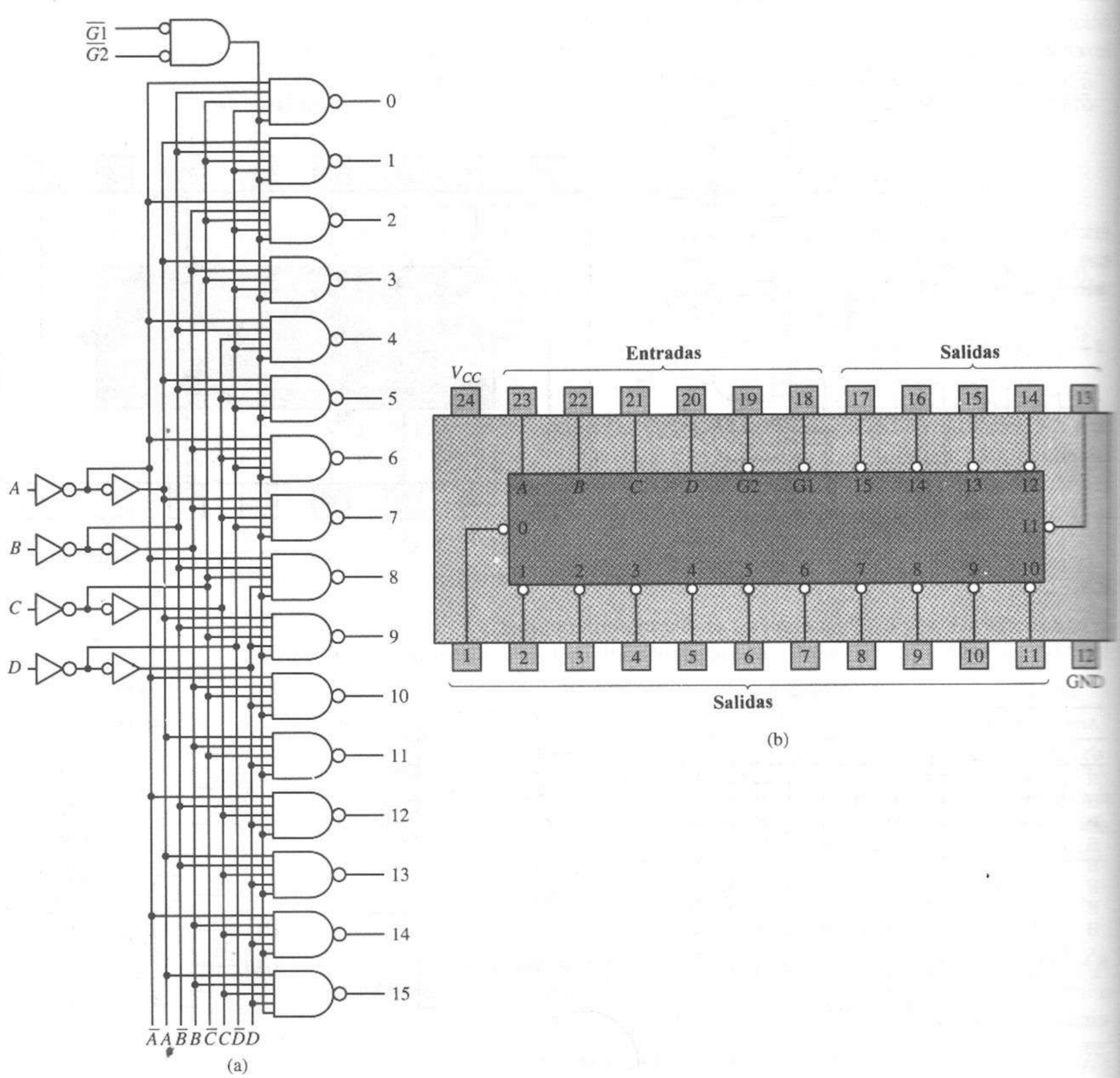


Figura 4.9 Módulo decodificador 74154. **(a)** Circuito lógico. **(b)** Configuración de *pines* del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

direcciones en las memorias de las computadoras y los sistemas de entrada/salida es tal vez uno de los más importantes. En esta aplicación, cada uno de 2^n dispositivos (celdas de memoria o puertos de entrada/salida) tiene asignado un único número binario de n bits, o *dirección*, que permite distinguirlo de los

Entradas						Salidas															
$\overline{G1}$	$\overline{G2}$	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
H	L	×	×	×	×	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	H	×	×	×	×	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	×	×	×	×	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

(c)

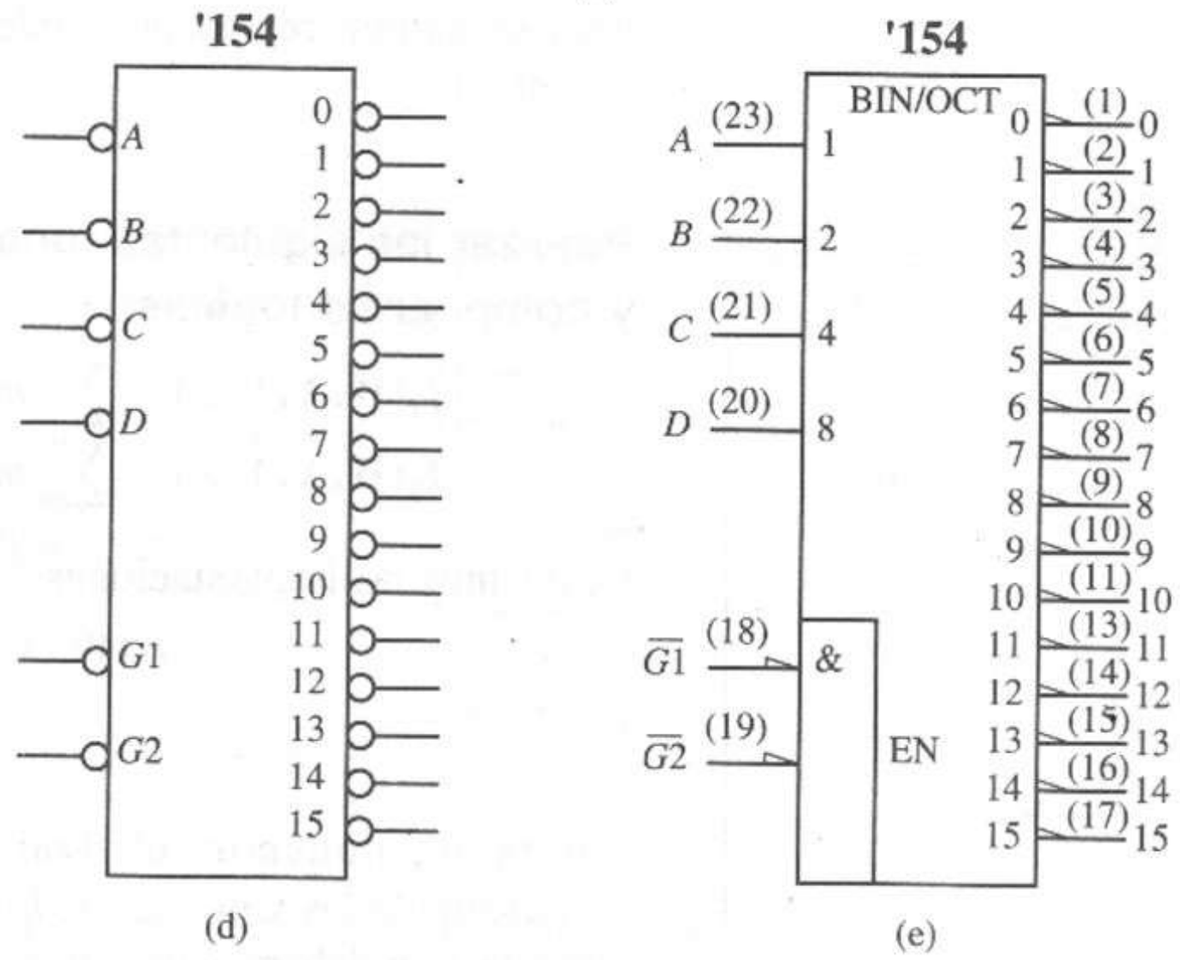


Figura 4.9 (continuación) Módulo decodificador 74154. **(c)** Tabla funcional. **(d)** Símbolo genérico. **(e)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

demás dispositivos. Una computadora designa un dispositivo específico para que intervenga en una operación transmitiendo su dirección por n líneas de señal. Como muestra la figura 4.10, un decodificador n a 2^n decodifica la dirección de n bits activando una de 2^n líneas de selección para tener acceso a uno de los dispositivos. Por ejemplo, en la memoria de una computadora, cada dirección correspondería a un grupo de bits (dígitos binarios) de información almacenado en la memoria. En una memoria sencilla de 4K ($1K = 2^{10} = 1024$), donde $n = 12$, se necesita un total de 4096 líneas de selección.

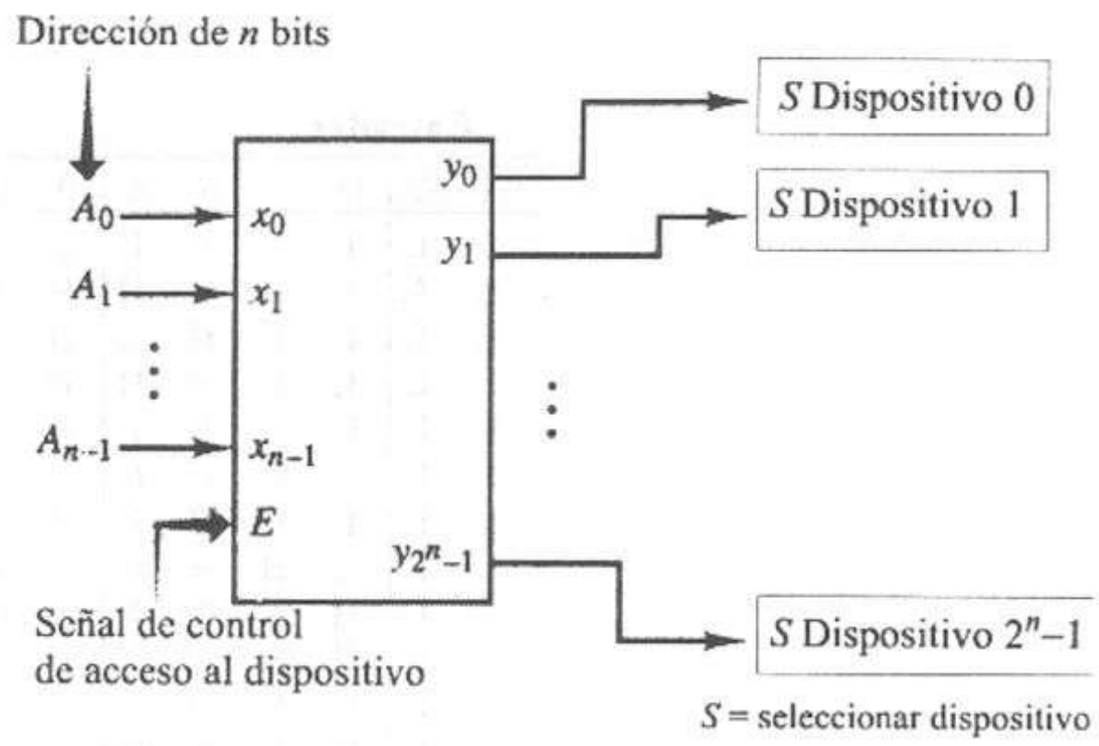


Figura 4.10 Decodificación de una dirección en un sistema digital.

Generación de mintérminos

Los decodificadores n a 2^n también tienen muchas aplicaciones en la síntesis de las redes digitales de conmutación, actuando como generadores de mintérminos, pues las salidas representan todos los mintérminos posibles de las n variables de entrada.

EJEMPLO 4.2

Realizar las siguientes funciones usando un 74154 y compuertas lógicas:

$$f_1(W,X,Y,Z) = \sum m(1,9,12,15)$$
$$f_2(W,X,Y,Z) = \sum m(0,1,2,3,4,5,7,8,10,11,12,13,14,15)$$

Utilizamos las implantaciones 2 y 3 del ejemplo 4.1:

$$f_1(W, X, Y, Z) = \overline{\bar{m}_1 \bar{m}_9 \bar{m}_{12} \bar{m}_{15}}$$

y

$$f_2(W, X, Y, Z) = \bar{m}_6 \cdot \bar{m}_9$$

Por tanto, podemos utilizar el 74154 para generar los mintérminos complementados y el 7420 y 7408 para generar f_1 y f_2 como en la figura 4.11. Observe que debemos unir $W = D$, $X = C$, $Y = B$ y $Z = A$.

Decodificadores BCD a decimal

Con frecuencia, los circuitos lógicos combinatorios se utilizan para convertir datos codificados en un esquema a otro formato. Se usan convertidores de código para convertir BCD a decimal, exceso 3 a decimal, binario a exceso 3, etcétera. Por ejemplo, la 4.12a muestra un decodificador BCD a decimal, y la figura 4.12b muestra los códigos BCD y los dígitos correspondientes. Este decodificador es similar al decodificador binario 4 a 16 ya descrito, pero con sólo 10 salidas, una por cada dígito decimal.

Para diseñar el decodificador BCD a decimal, podemos trazar un mapa K para cada una de las 10 salidas y deducir su ecuación lógica. Cada mapa K contiene exactamente un mintérmino, correspondiente al número decimal de esa salida, y seis términos prescindibles, 10 a 15, pues estos números no existen

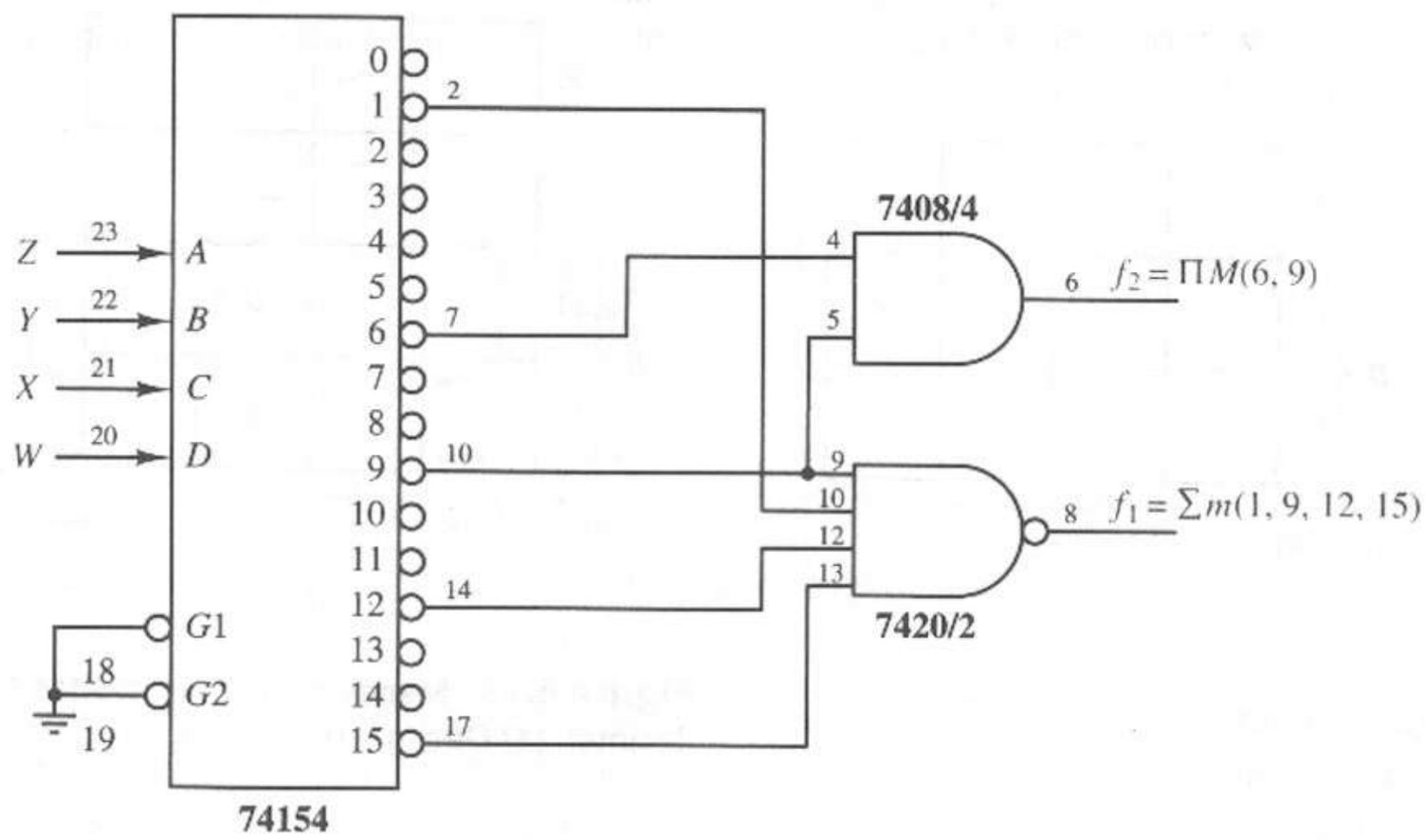
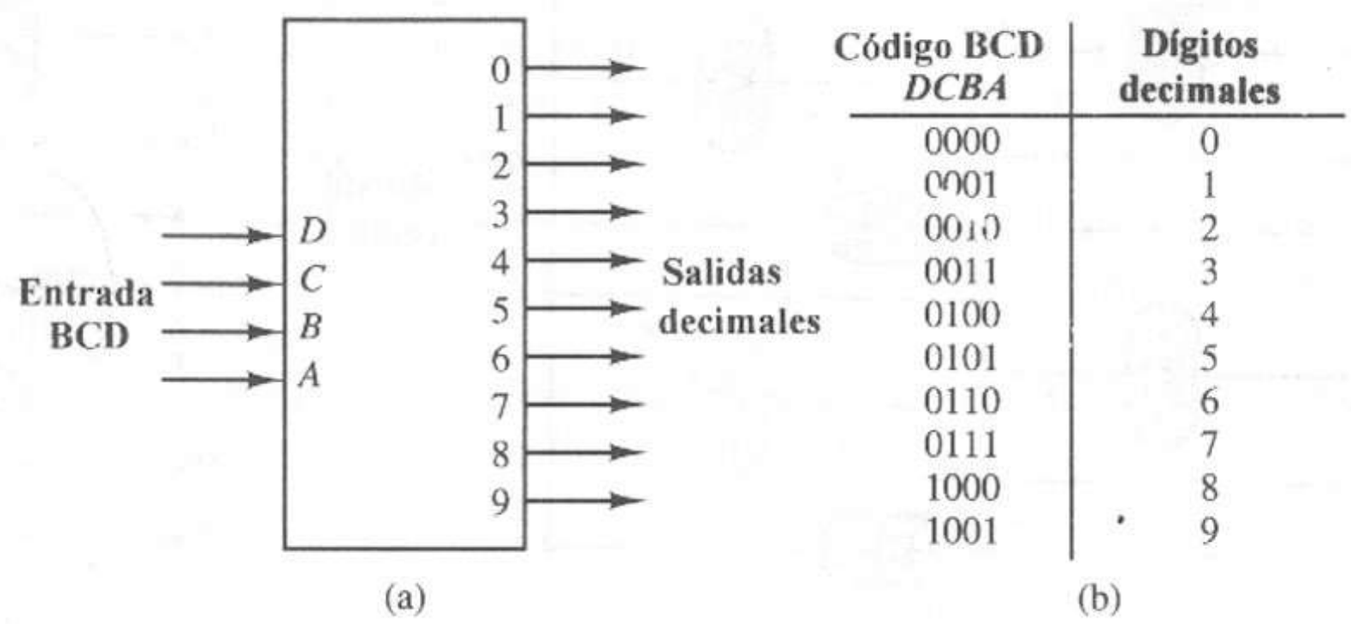


Figura 4.11 Realización de funciones de conmutación mediante un decodificador.



Código BCD *DCBA*	Dígitos decimales
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Figura 4.12 Decodificador BCD a decimal. **(a)** Símbolo lógico. **(b)** Códigos BCD y dígitos decimales.

en el código BCD. La figura 4.13 muestra tres de estos mapas K. El conjunto completo de ecuaciones lógicas que describe al decodificador BCD es:

Decimal 0	$\bar{D}\bar{C}\bar{B}\bar{A}$	Decimal 5	$C\bar{B}A$
1	$\bar{D}\bar{C}\bar{B}A$	6	$CB\bar{A}$
2	$\bar{C}B\bar{A}$	7	CBA
3	$\bar{C}BA$	8	$D\bar{A}$
4	$C\bar{B}\bar{A}$	9	DA

Puesto que los códigos 0 a 9 tienen los mismos códigos en binario y en BCD, también podemos utilizar un decodificador binario 4 a 16 74154 para implantar un decodificador BCD a decimal empleando sólo las salidas $\overline{m}_0$ a $\overline{m}_9$ e ignorando las salidas $\overline{m}_{10}$ a $\overline{m}_{15}$. El 74154 podría ser más caro que una realización mínima

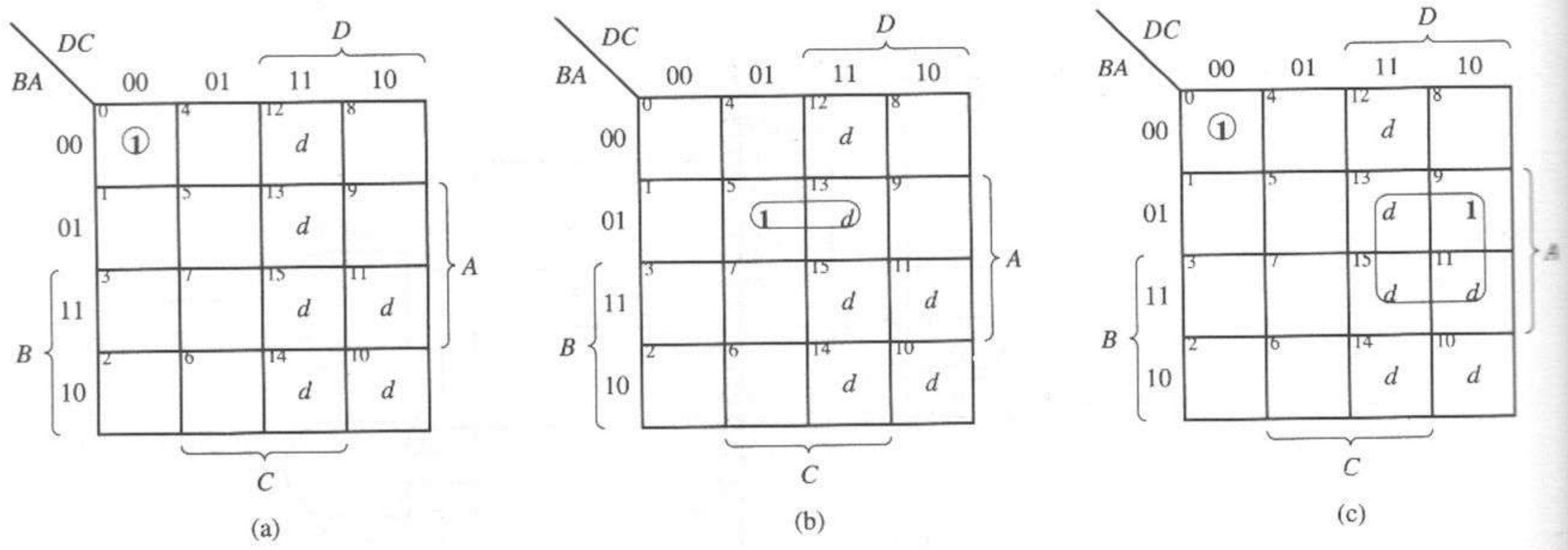


Figura 4.13 Mapas K para las salidas 0, 5 y 9 de un convertidor de código BCD a decimal. **(a)** Decimal 0 = $\overline{D}\,\overline{C}\,\overline{B}\,\overline{A}$. **(b)** Decimal 5 = $C\overline{B}A$. **(c)** Decimal 9 = DA.

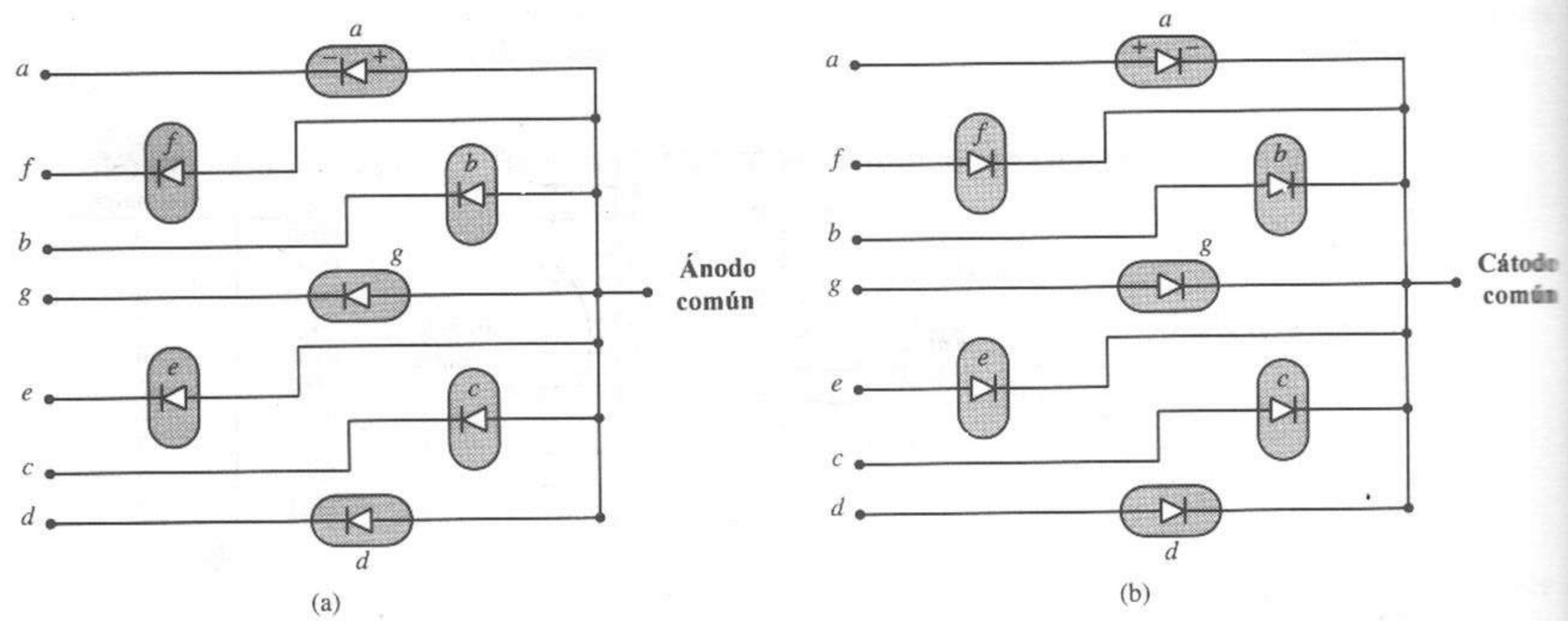


Figura 4.14 Elementos de un display de 7 segmentos. **(a)** Ánodo común (MAN 72A). **(b)** Cátodo común (MAN 74A).

de las ecuaciones lógicas para el decodificador BCD aquí expuesto, pero podría ser una solución conveniente si está disponible.

Decodificadores de display

Otra aplicación común de la decodificación es la conversión de datos codificados a un formato adecuado para el control de un *display* numérico. Por ejemplo, los relojes digitales y otros equipos electrónicos exhiben con frecuencia dígitos decimales codificados en BCD en *displays* de siete segmentos. Los *displays LED de siete segmentos* tienen siete diodos emisores de luz (LED) dispuestos

como en la figura 4.14. Ciertas combinaciones de los LED se iluminan para crear dígitos numéricos y otros símbolos. Por ejemplo, los segmentos que normalmente se activan para mostrar los dígitos decimales aparecen en la figura 4.15.

Figura 4.15 Dígitos decimales exhibidos mediante los elementos de un *display* de siete segmentos.

Un LED emite luz cuando el voltaje en su ánodo es suficientemente mayor que el voltaje en su cátodo. Esto se ilustra en el diodo *a* de las figuras 4.14a y b, en donde el ánodo se marca con "+" y el cátodo con "–". En los circuitos digitales, estos voltajes se crean aplicando un voltaje alto al ánodo y uno bajo al cátodo.

Para minimizar el número de señales de control, los ánodos de los LED se conectan por lo general a un punto común, llamado *ánodo común*, como se muestra en la figura 4.14a, o bien, los cátodos se conectan a un *cátodo común*, como en la figura 4.14b. En la configuración con ánodo común, los ánodos se conectan por lo general a un voltaje alto y los cátodos se controlan en forma individual. En consecuencia, un 0 lógico aplicado a un cátodo ilumina ese LED, mientras que un voltaje de 1 lógico desactiva el LED. Se utilizan las condiciones opuestas en la configuración de cátodo común. Por tanto, las entradas a la configuración de ánodo común se pueden considerar activas bajas, ya que las señales bajas activan los LED, y las entradas a la configuración de cátodo común son activas altas.

A fin de exhibir un número codificado en formato BCD, podemos diseñar un decodificador que convierta los códigos BCD a los valores lógicos necesarios para los siete segmentos. Comenzamos creando una tabla de verdad que enumere los segmentos por activar para cada dígito decimal, como en la tabla 4.2. De esta tabla podemos obtener un mapa K para cada uno de los siete segmentos que nos servirá para deducir las ecuaciones lógicas mínimas. Por ejemplo, la figura 4.16 muestra los mapas K para los segmentos *a* y *b*, donde hemos deducido las expresiones de producto de sumas.

Podemos generalizar este procedimiento al diseño de circuitos lógicos que conviertan datos de un formato de código arbitrario a otro. Mediante el formato de tabla de verdad de la tabla 4.2, enumeramos todos los códigos de entrada como entradas al circuito, con los códigos de salida correspondientes como salidas del circuito. Después trazamos un mapa K y deducimos una ecuación lógica para cada salida, dejando que los códigos de entrada no especificados como condiciones prescindibles.

4.3 Codificadores

Un *codificador* es un módulo lógico combinatorio que asigna un código de salida único (un número binario) a cada señal de entrada aplicada al dispositivo;

TABLA 4.2 CONVERSIÓN DE CÓDIGO BCD A CÓDIGO DE SIETE SEGMENTOS

Dígito decimal	Código decimal				Segmentos del *display*						
	D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

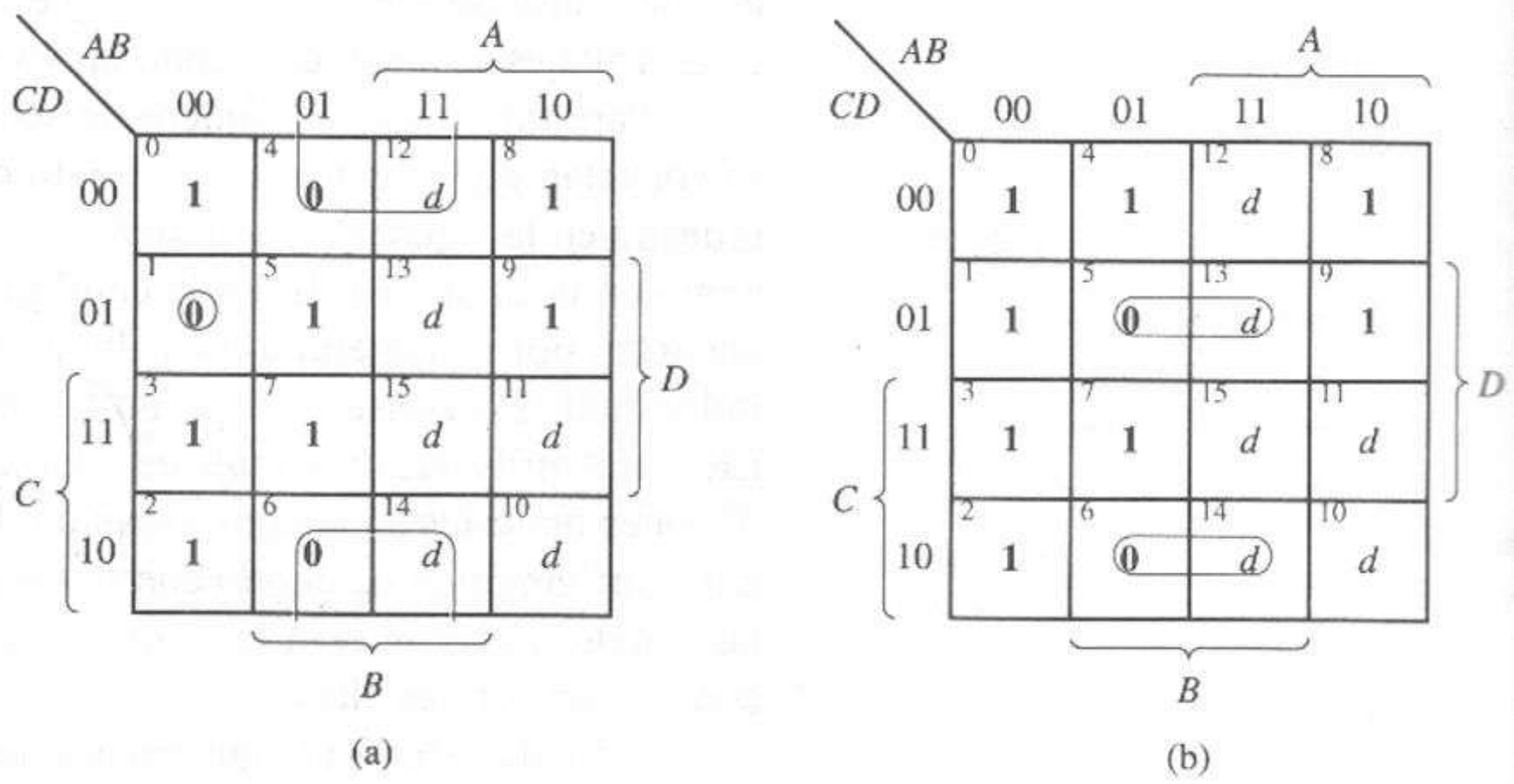


Figura 4.16 Mapas K para un convertidor de BCD a código de 7 segmentos. **(a)** Segmento $a = (\overline{B} + D)(A + B + C + \overline{D})$. **(b)** Segmento $b = (\overline{B} + C + \overline{D})(\overline{B} + \overline{C} + D)$.

como tal, es lo opuesto de un decodificador. Si un módulo codificador tiene n entradas, el número de salidas s debe satisfacer la expresión

$$2^s \geq n \tag{4.7}$$

o bien

$$s \geq \log_2 n$$

4.3.1 Estructuras de un circuito codificador

Codificadores con entradas mutuamente excluyentes

Consideremos primero el caso en que las entradas son mutuamente excluyentes; es decir, una y sólo una de las líneas de entrada está activa en un instante en

particular; nunca están activas al mismo tiempo dos o más líneas de entrada. En este caso, las combinaciones de entradas que nunca ocurren se pueden utilizar como condiciones prescindibles.

EJEMPLO 4.3

Diseñar un codificador para cuatro líneas de entrada si una y sólo una está activa en cualquier instante. Ver la figura 4.17a.

Definamos el código:

	A_1	A_0
$X_0 \rightarrow$	0	0
$X_1 \rightarrow$	0	1
$X_2 \rightarrow$	1	0
$X_3 \rightarrow$	1	1

Las funciones de salida proporcionan el valor binario del subíndice de la variable de entrada. Las figuras 4.17b y c muestran la tabla de verdad y los mapas K para el codificador. De los mapas K vemos que

$$A_1 = X_3 + X_2$$
$$A_0 = X_3 + X_1$$

La figura 4.17d muestra el diagrama del circuito lógico para el codificador.

El codificador anterior requiere que una (y sólo una) de las entradas esté activa en todo momento, una condición única. Supongamos que relajamos las restricciones un poco y que diseñamos un codificador que permita la ocurrencia de todas las combinaciones de entradas, pero cuya salida sea un código distinto de cero sólo si una de las líneas de entrada está activa.

EJEMPLO 4.4

Diseñar un codificador de cuatro líneas cuya salida es un código distinto de cero a menos que una y sólo una de las líneas de entrada esté activa.

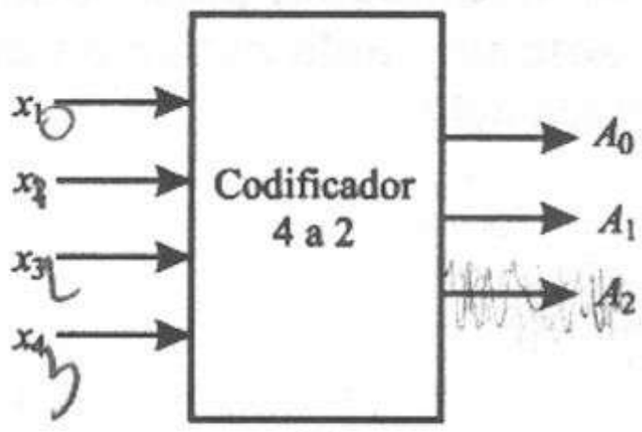
Definamos el código:

	A_2	A_1	A_0
$X_1 \rightarrow$	0	0	1
$X_2 \rightarrow$	0	1	0
$X_3 \rightarrow$	0	1	1
$X_4 \rightarrow$	1	0	0
Los demás $\rightarrow$	0	0	0

Este codificador también tiene como salida el subíndice de la línea de entrada activa. Sin embargo, su salida es el código nulo si ninguna línea de entrada está activa o si varias líneas están activas. La figura 4.18 detalla el diseño:

$$A_2 = X_4\bar{X}_3\bar{X}_2\bar{X}_1$$
$$A_1 = \bar{X}_4\bar{X}_3X_2\bar{X}_1 + \bar{X}_4X_3\bar{X}_2\bar{X}_1$$
$$A_0 = \bar{X}_4\bar{X}_3\bar{X}_2X_1 + \bar{X}_4X_3\bar{X}_2\bar{X}_1$$

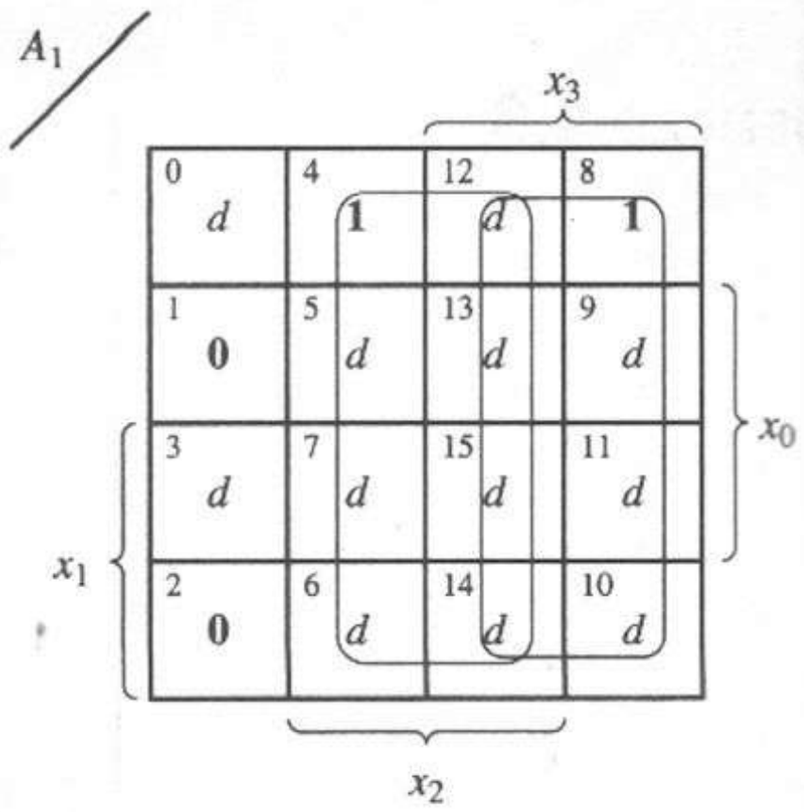
Observe que la lógica necesaria para implantar el segundo codificador es más compleja que para el primero.

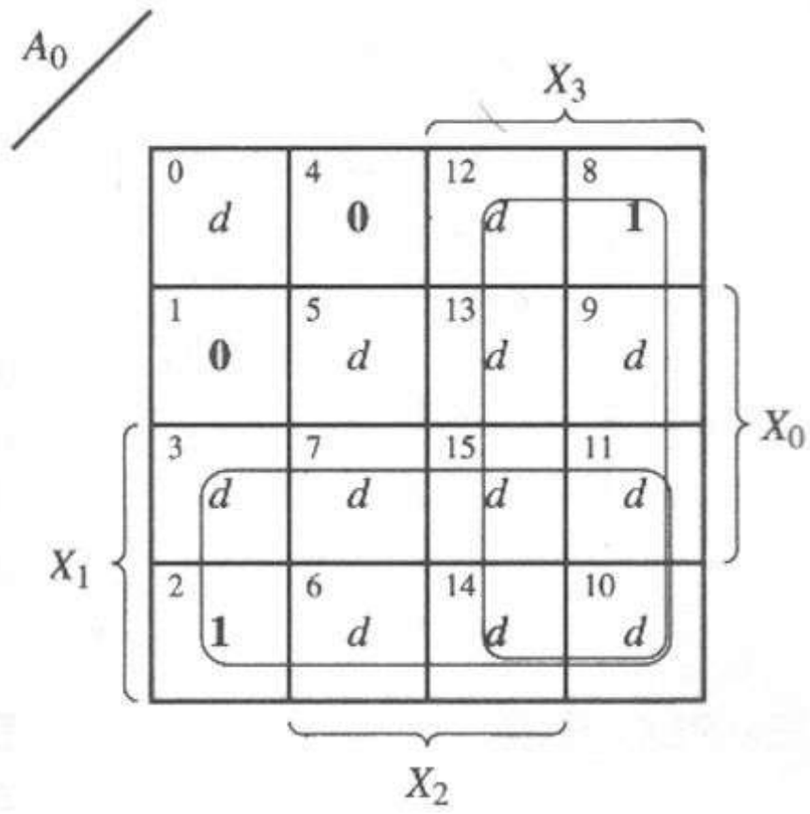


(a)

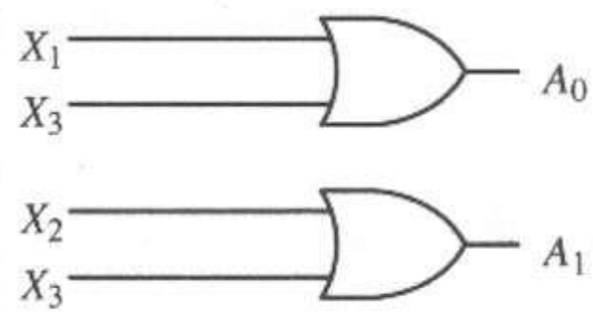
X_3	X_2	X_1	X_0	A_1	A_0
0	0	0	0	d	d
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	d	d
0	1	0	0	1	0
0	1	0	1	d	d
0	1	1	0	d	d
0	1	1	1	d	d
1	0	0	0	1	1
1	0	0	1	d	d
1	0	1	0	d	d
1	0	1	1	d	d
1	1	0	0	d	d
1	1	0	1	d	d
1	1	1	0	d	d
1	1	1	1	d	d

(b)



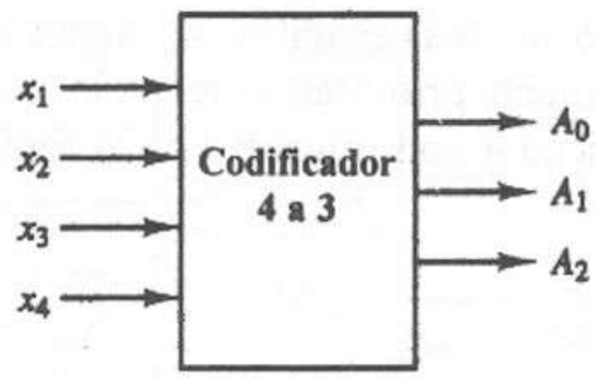


(c)



(d)

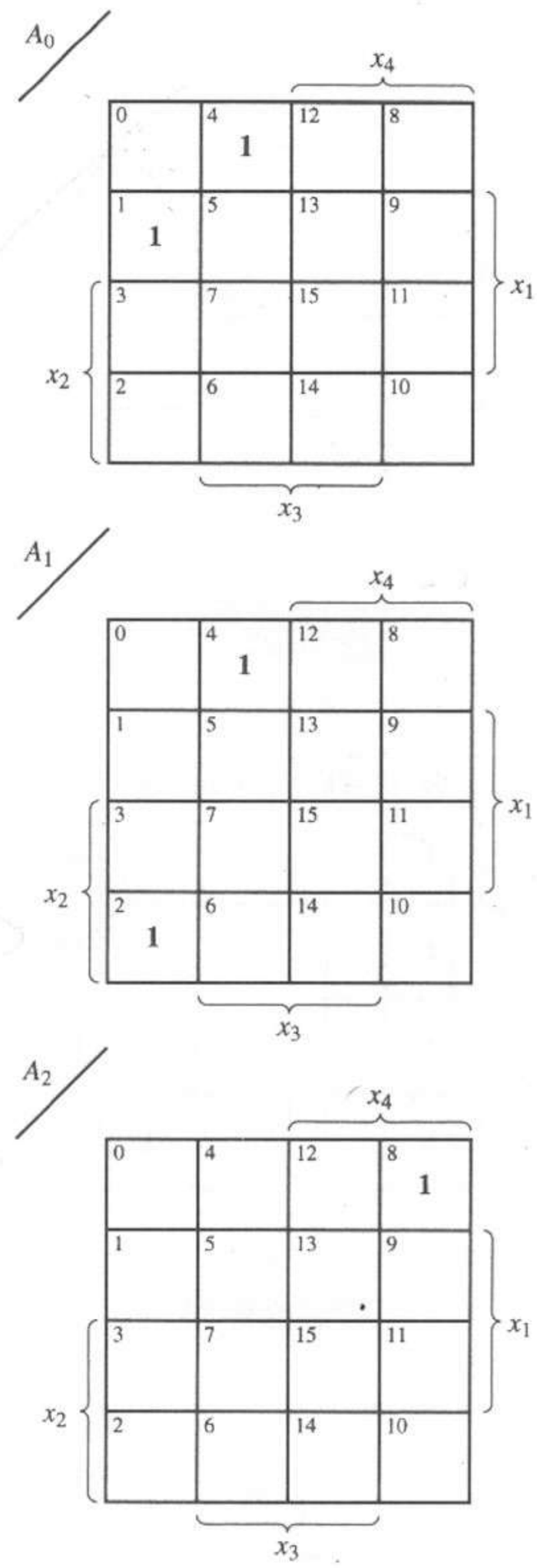
Figura 4.17 Codificador cuatro a dos líneas. **(a)** Diagrama funcional. **(b)** Tabla de verdad. **(c)** Mapas K. **(d)** Diagrama lógico.



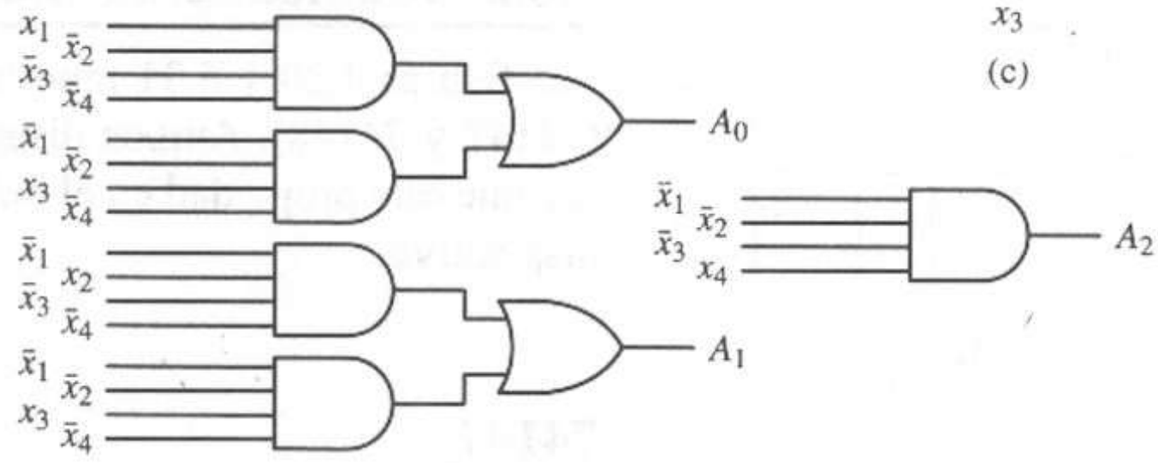
(a)

X_4	X_3	X_2	X_1	A_2	A_1	A_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

(b)



(c)



(d)

Figura 4.18 Codificador cuatro a tres líneas. **(a)** Diagrama funcional. **(b)** Tabla de verdad. **(c)** Mapas K. **(d)** Diagrama lógico.

Codificadores con prioridad

Otro tipo de codificador es el *codificador con prioridad*. Este codificador permite que varias líneas de entrada estén activas al mismo tiempo y envía el valor binario del subíndice de la línea de entrada con mayor prioridad. Para que el diseño sea más sencillo, se asigna la máxima prioridad al subíndice más alto, la siguiente prioridad al segundo subíndice más alto, etcétera. Consideremos el codificador con prioridad de la figura 4.19. Las líneas de entrada se codifican como

	A_1	A_0
$X_0 \rightarrow$	0	0
$X_1 \rightarrow$	0	1
$X_2 \rightarrow$	1	0
$X_3 \rightarrow$	1	1

Si ninguna línea de entrada está activa, el codificador con prioridad produce $(A_1A_0) = (00)$. Si sólo está activa una línea, el codificador produce el valor binario del subíndice de la línea activa. Si está activa más de una entrada, el codificador envía el valor binario del máximo subíndice de las líneas activas. La figura 4.19b muestra la tabla de verdad para el codificador. Observe que las dos líneas de salida adicionales indican que ninguna línea está activa ($EO = 1$) y que una o más entradas están activas ($GS = 1$). Las figuras 4.19c y d presentan el mapa K y el diagrama lógico de la función, que se reduce a

$$A_1 = X_2 + X_3 \tag{4.8}$$

$$A_0 = X_3 + X_1\bar{X}_2 \tag{4.9}$$

y

$$EO = \overline{GS} = \overline{X_3 + X_2 + X_1 + X_0} \tag{4.10}$$

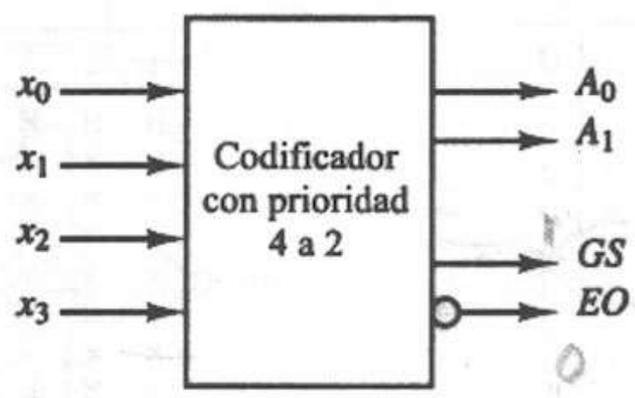
Las dos funciones de salida A_1 y A_0 son independientes de X_0. Observe que el codificador con prioridad puede realizar la tabla de verdad de la figura 4.17b. En consecuencia, el codificador con prioridad también puede funcionar como codificador de mintérminos.

4.3.2 Codificadores MSI estándar

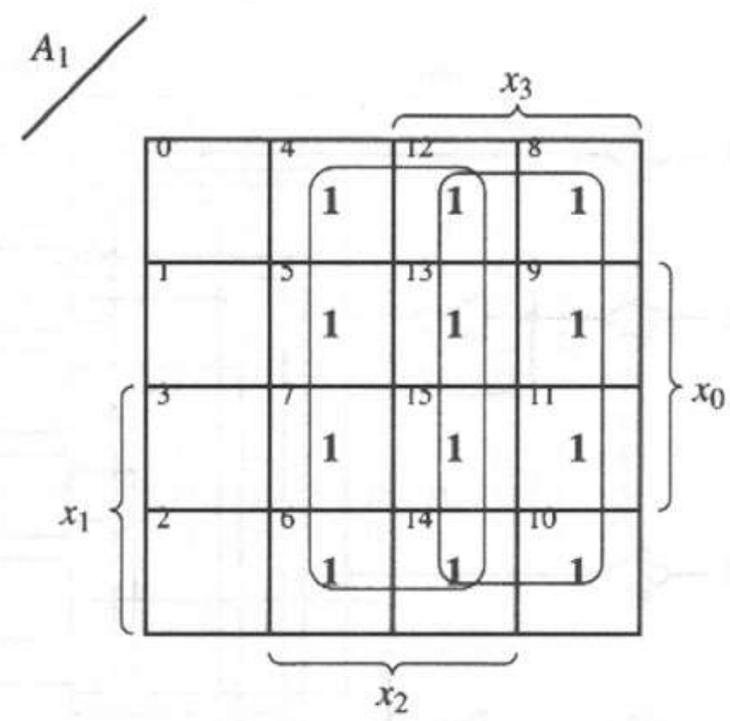
Las figuras 4.20 y 4.21 muestran dos codificadores con prioridad modulares (74147 y 74148). Ambos dispositivos tienen entradas y salidas activas bajas. Busque esta propiedad en el diagrama lógico y en la tabla de función para cada dispositivo.

74147

El 74147, cuyo circuito lógico aparece en la figura 4.20a, toma 10 líneas (0, 1,..., 9) y las codifica en 4 líneas (D, C, B, A), como en la tabla de función de la figura 4.20b. Observe que la línea de entrada 0 no está conectada al circuito, como lo sugieren las ecuaciones 4.8 y 4.9. El arreglo de *pines* de un paquete dual en línea del 74147 aparece en la figura 4.20c.



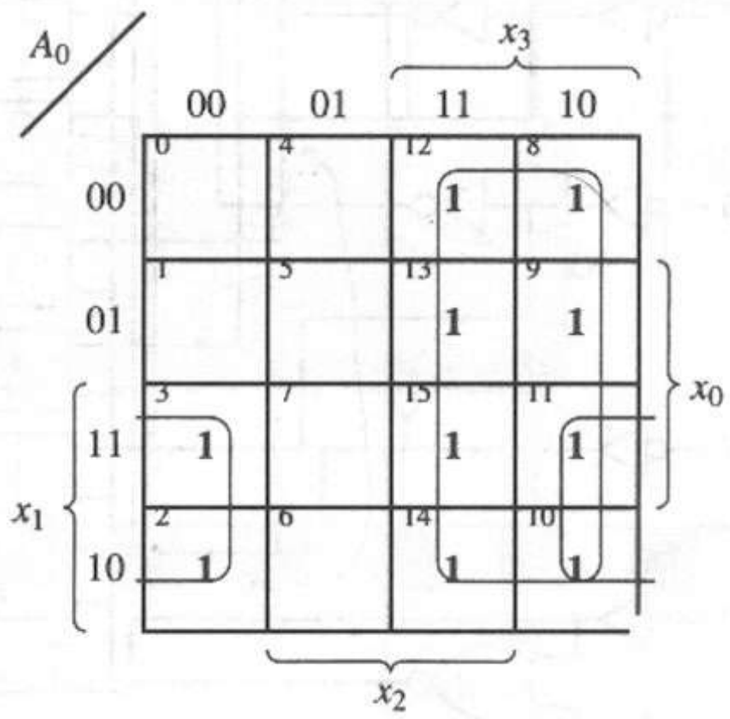
(a)



$A_1 = X_2 + X_3$

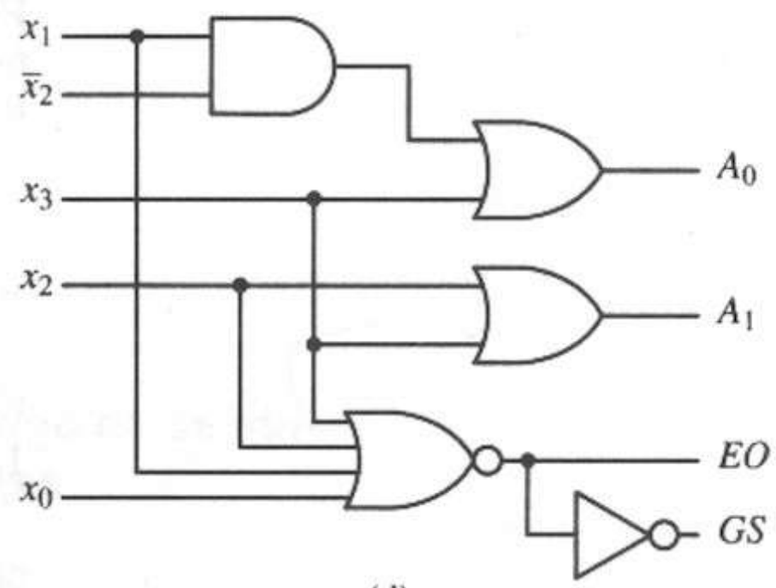
Entradas				Salidas			
X_3	X_2	X_1	X_0	A_1	A_0	GS	EO
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	0

(b)



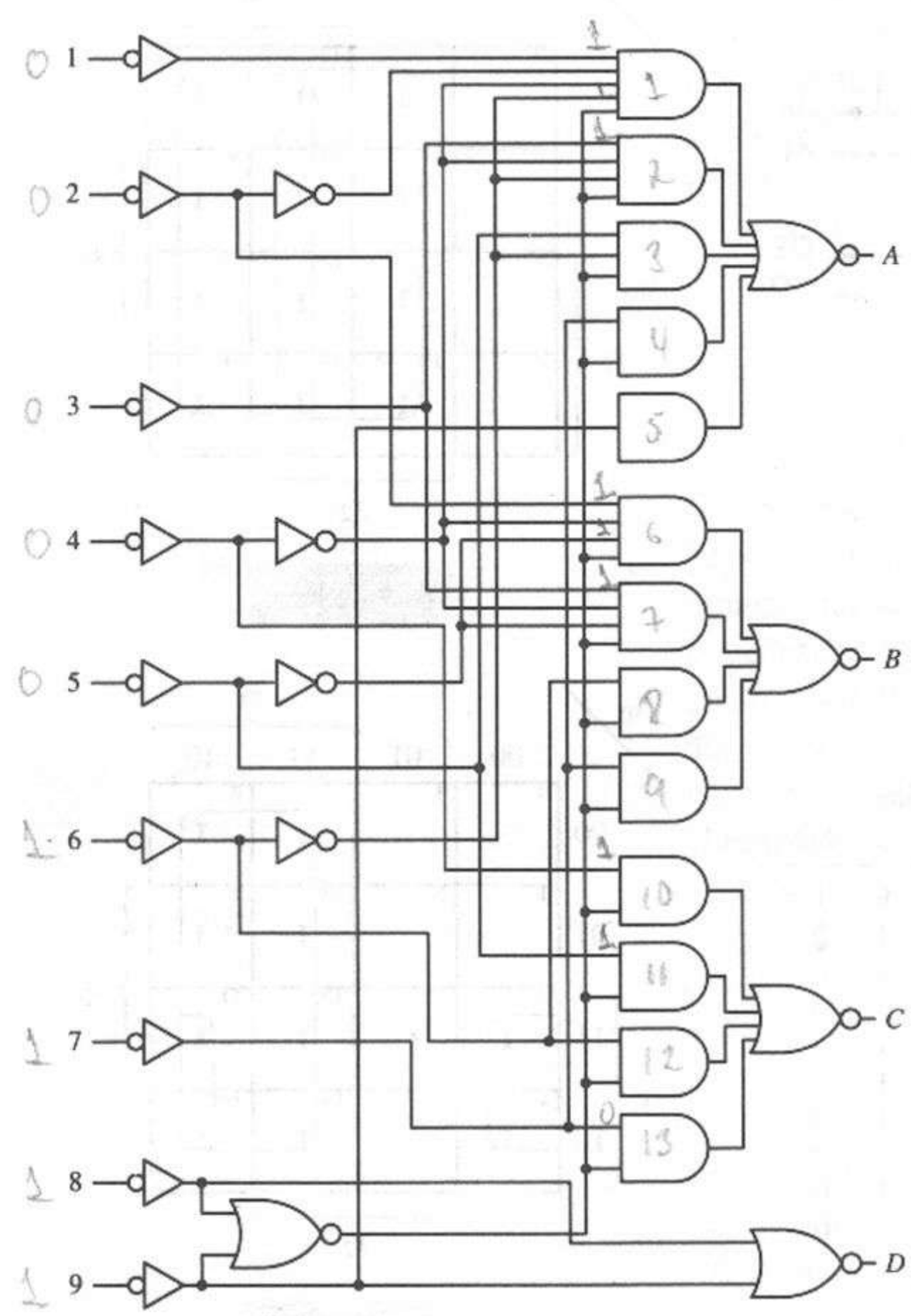
$A_0 = X_3 + X_1\overline{X_2}$

(c)



(d)

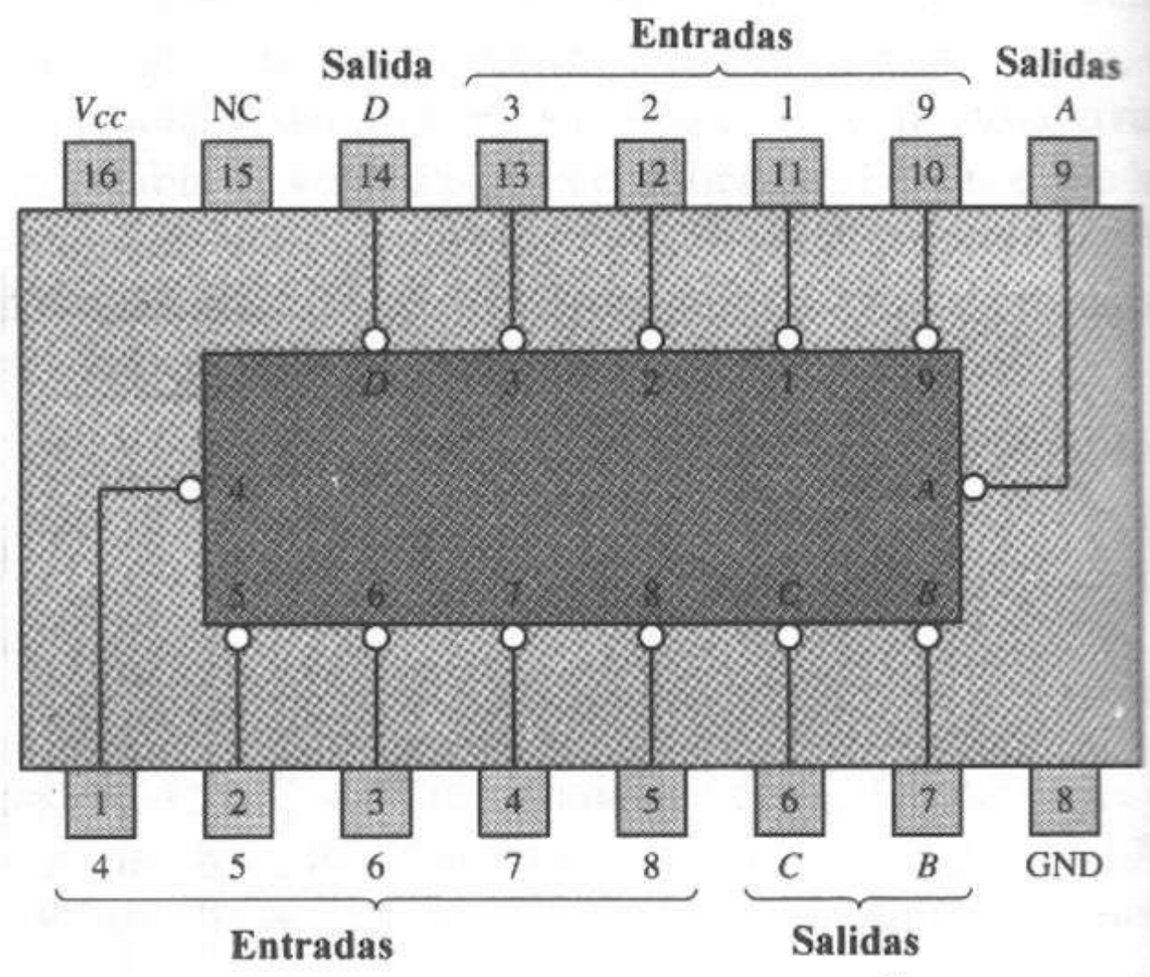
Figura 4.19 Codificador con prioridad de cuatro a dos líneas. **(a)** Diagrama funcional. **(b)** Tabla de verdad. **(c)** Mapas K. **(d)** Diagrama lógico.



(a)

Entradas									Salidas			
1	2	3	4	5	6	7	8	9	*D*	*C*	*B*	*A*
H	H	H	H	H	H	H	H	H	H	H	H	H
×	×	×	×	×	×	×	×	L	L	H	H	L
×	×	×	×	×	×	×	L	H	L	H	H	H
×	×	×	×	×	×	L	H	H	H	L	L	L
×	×	×	×	×	L	H	H	H	H	L	L	H
×	×	×	×	L	H	H	H	H	H	L	H	L
×	×	×	L	H	H	H	H	H	H	L	H	H
×	×	L	H	H	H	H	H	H	H	H	L	L
×	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

(b)

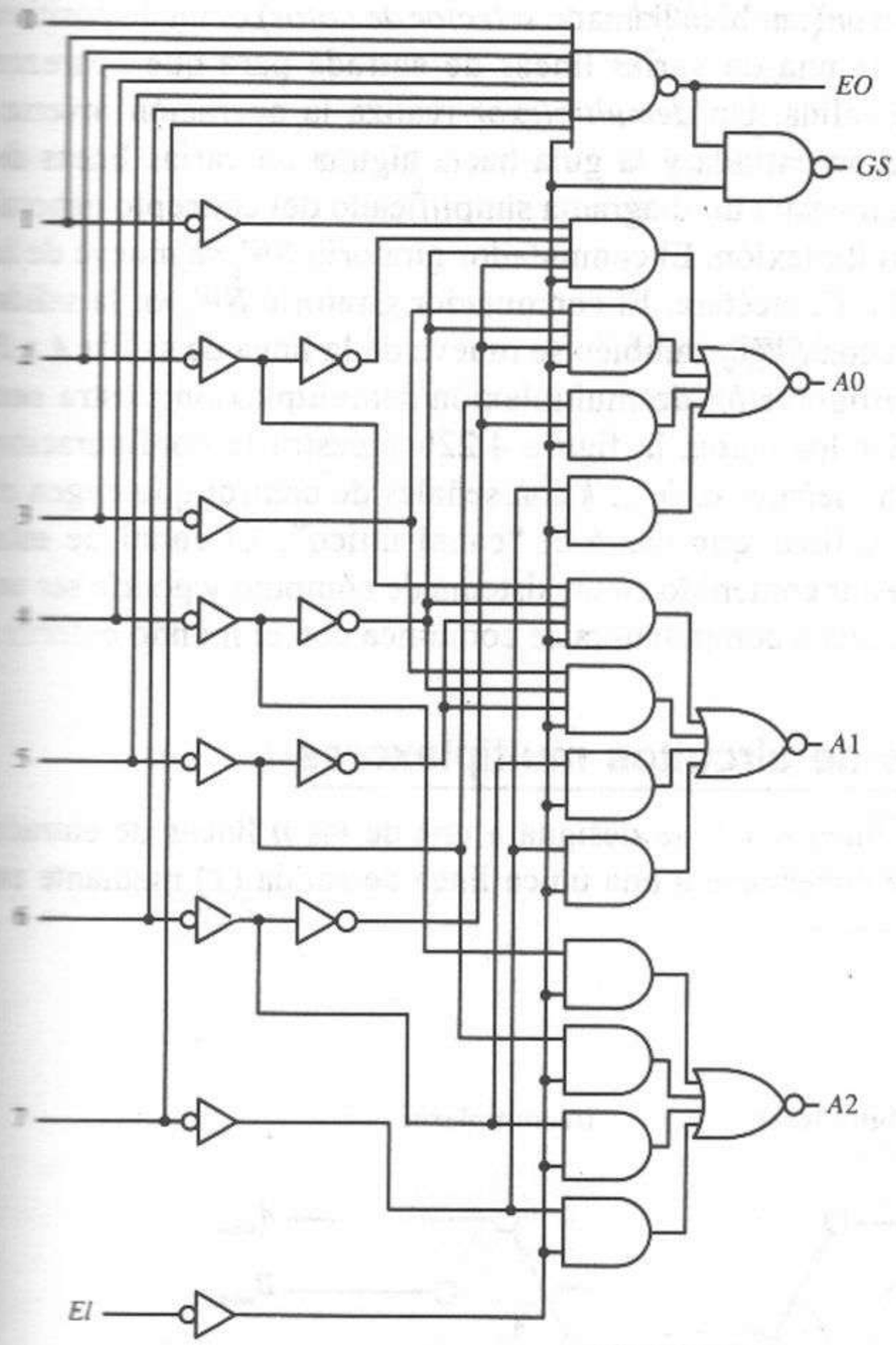


(c)

Figura 4.20 Módulo codificador con prioridad 74147. **(a)** Diagrama lógico. **(b)** Tabla funcional. **(c)** Configuración del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

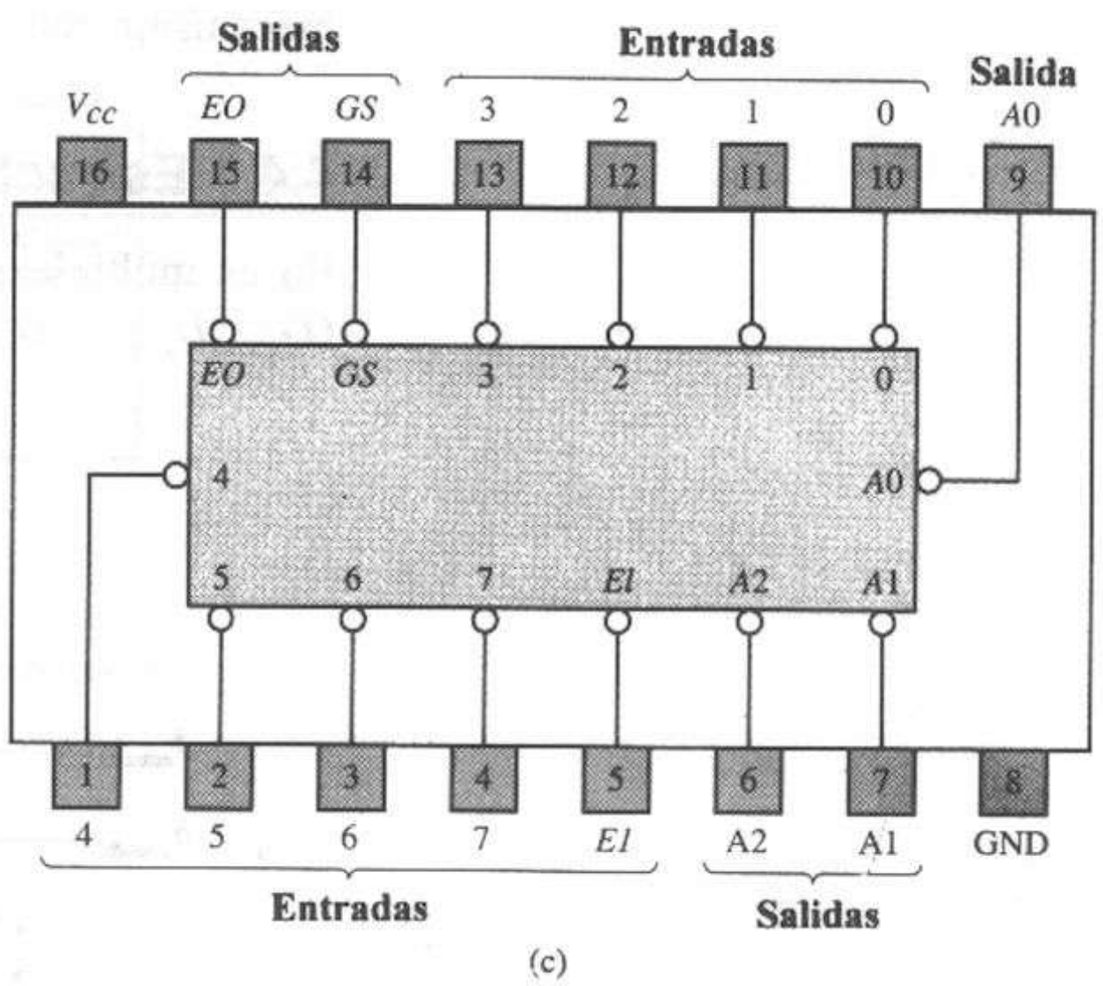
74148

El circuito lógico 74148, que aparece en la figura 4.21a, toma ocho líneas (0, 1,..., 7) y las codifica en tres líneas (*A*2, *A*1, *A*0) según la tabla funcional de la figura 4.21b. Una señal de habilitación de entrada *EI* se conecta a todas las compuertas lógicas de primer nivel para controlar su operación; cuando *EI* está activa (baja), el circuito opera. El 74148 también tiene dos señales de salida adicionales, *EO* y *GS*. *EO* está activa (baja) cuando ninguna de las líneas de entrada está activa. *GS* está activa (baja) cuando una o más líneas de entrada están activas. El arreglo de *pines* de un paquete dual en línea del 74148 aparece en la figura 4.21c.



Entradas									Salidas				
EI	0	1	2	3	4	5	6	7	*A2*	*A1*	*A0*	*GS*	*EO*
H	×	×	×	×	×	×	×	×	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	×	×	×	×	×	×	×	L	L	L	L	L	H
L	×	×	×	×	×	×	L	H	L	L	H	L	H
L	×	×	×	×	×	L	H	H	L	H	L	L	H
L	×	×	×	×	L	H	H	H	L	H	H	L	H
L	×	×	×	L	H	H	H	H	H	L	L	L	H
L	×	×	L	H	H	H	H	H	H	L	H	L	H
L	×	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

(b)



(c)

Figura 4.21 Módulo codificador con prioridad 74148. **(a)** Diagrama lógico. **(b)** Tabla funcional. **(c)** Configuración del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

EJEMPLO 4.5

Para el 74148, decir cuál es el código de salida (*EO*, *GS*, *A*2, *A*1, *A*0) en las siguientes condiciones:

$$(EI, 7, 6, 5, 4, 3, 2, 1, 0) = (0, 1, 0, 1, 0, 1, 0, 1, 1)$$

Como el dispositivo está habilitado y tres líneas están activas (las entradas 6, 4 y 2), *GS* está activa baja y *A*2, *A*1, *A*0 codifican la línea 6 (001):

$$(EO, GS, A2, A1, A0) = (1, 0, 0, 0, 1)$$

4.4 Multiplexores/selectores de datos

En general, un *multiplexor* (también llamado *selector de datos*) es un dispositivo modular que selecciona una de varias líneas de entrada para que aparezca en una única línea de salida. Un *demultiplexor* realiza la operación inversa: acepta una única línea de entrada y la guía hacia alguna de varias líneas de salida. La figura 4.22a muestra un diagrama simplificado del concepto general de multiplexión y demultiplexión. El conmutador giratorio SW_1 se mueve de la línea de entrada A a B a C, etcétera. El conmutador giratorio SW_2 en la salida del canal se sincroniza con SW_1 y también se mueve de la línea de salida A a B a C, etcétera. Esta configuración de multiplexión/demultiplexión ilustra una forma de elegir y guiar los datos; la figura 4.22b muestra la configuración lógica. En este caso, las señales $a, b,..., k$ son señales de control que eligen el conjunto de entradas/salidas que usará el "canal único". El canal de esta configuración podría estar contenido en un sistema de cómputo y podría ser un mecanismo con el que dicha computadora se comunica con el mundo exterior.

4.4.1 Estructuras de circuitos multiplexores

En un multiplexor de línea n a 1, se designa a una de las n líneas de entrada (D_{n-1}, D_{n-2},..., D_0) para conectarse a una única línea de salida (Y) mediante un

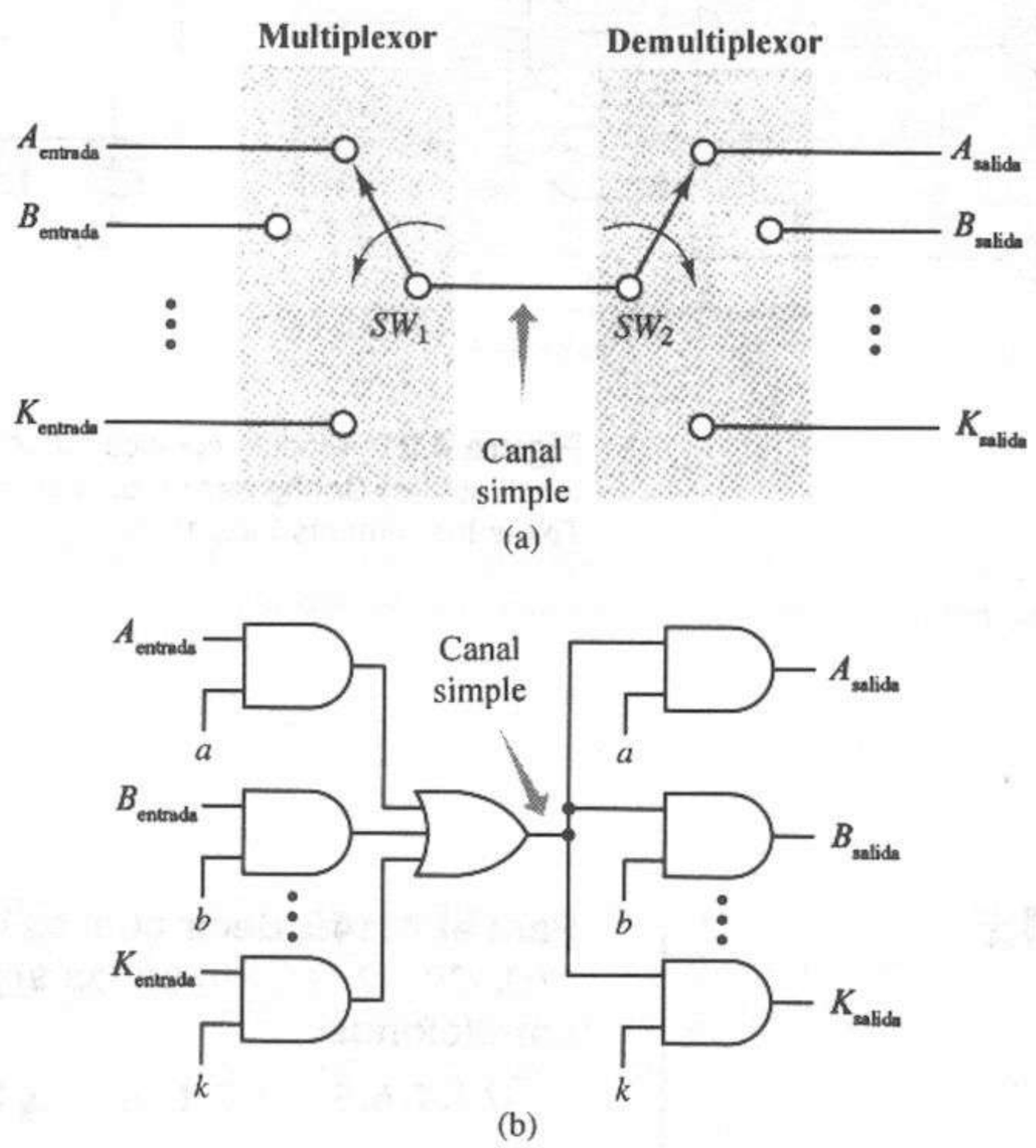


Figura 4.22 Sistema multiplexor/demultiplexor de K canales. **(a)** Operación de multiplexión/demultiplexión. **(b)** Configuración lógica sencilla para (a).

código de selección (S_{k-1},..., S_0), donde $n = 2^k$. Examinemos la figura 4.23a, que muestra un multiplexor de línea 4 a 1, con $B = S_1$ y $A = S_0$. El circuito conectará la línea de datos D_i a la salida Y cuando apliquemos el código

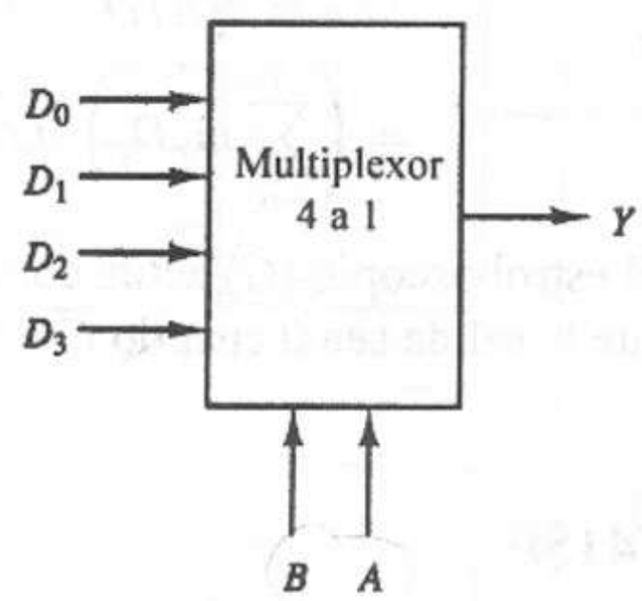
$$i = (BA)_2 \tag{4.11}$$

a las terminales de selección. La figura 4.23b muestra la tabla de verdad del multiplexor. Según esta tabla, podemos escribir

$$Y = (\bar{B}\bar{A})D_0 + (\bar{B}A)D_1 + (B\bar{A})D_2 + (BA)D_3 \tag{4.12}$$

El código de selección forma los mintérminos de dos variables, B y A. Por tanto, podemos escribir

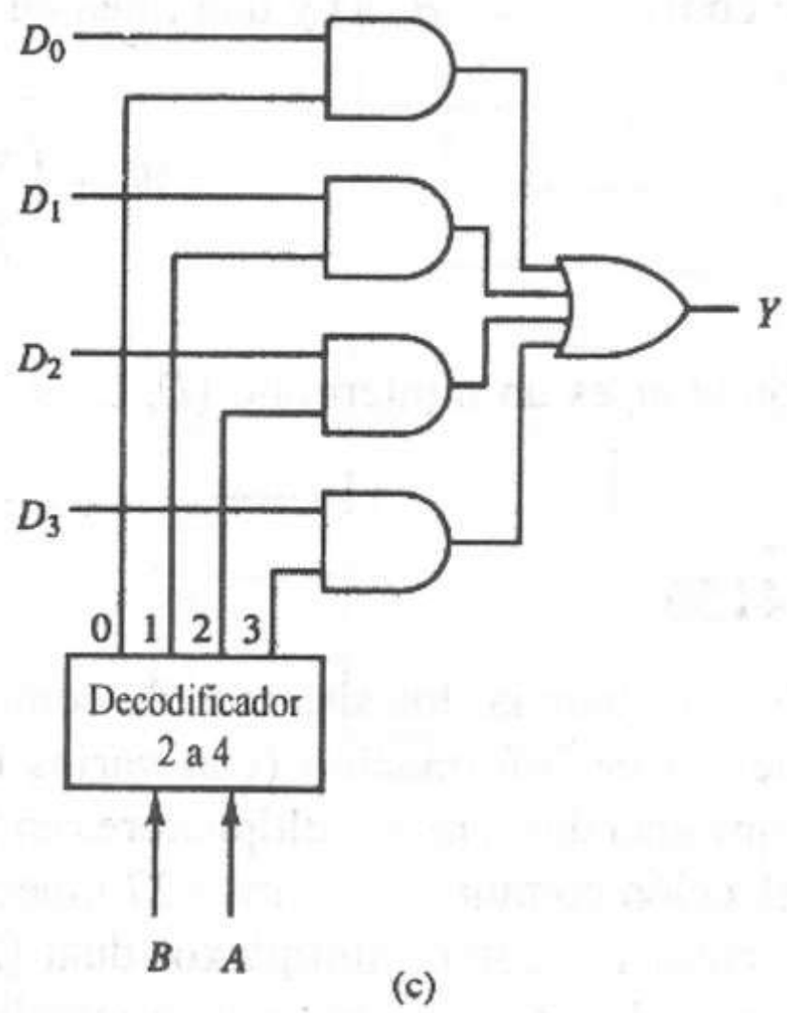
$$Y = \sum_{i=0}^{3} m_i D_i \tag{4.13}$$



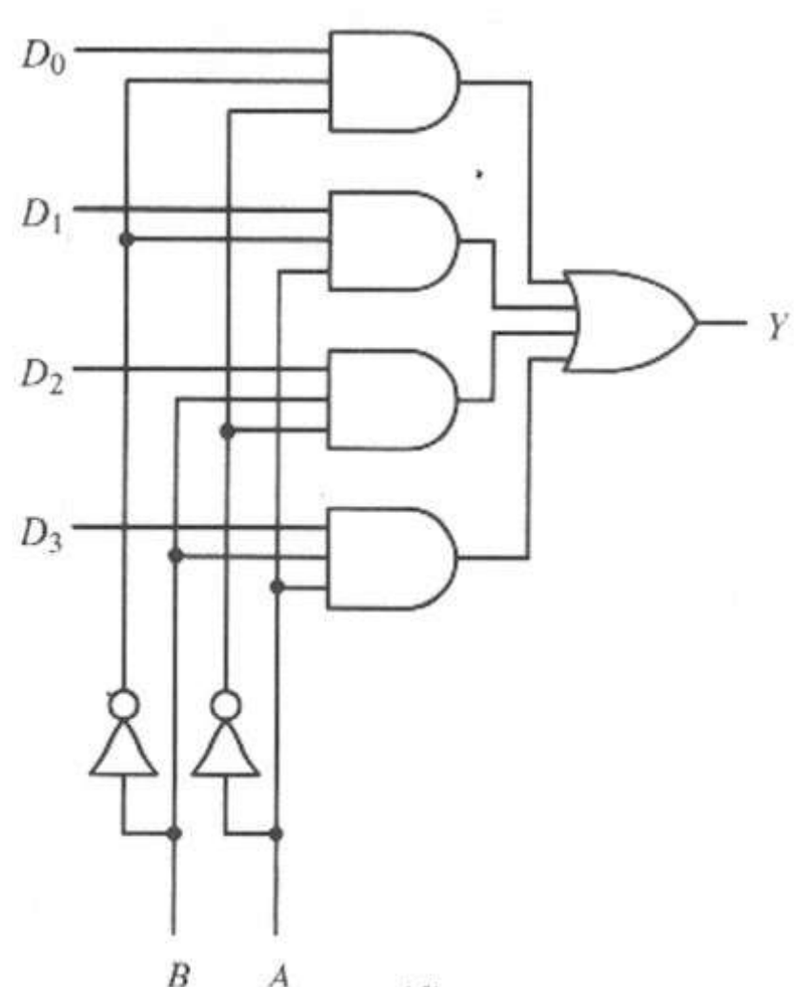
(a)

B	A	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

(b)



(c)



(d)

Figura 4.23 Diseño de multiplexor cuatro a uno. **(a)** Diagrama funcional. **(b)** Tabla de verdad. **(c)** Diagrama lógico. **(d)** Circuito equivalente de dos niveles.

donde los m_i son los mintérminos del código de selección. La figura 4.23c muestra el diagrama lógico del multiplexor 4 a 1. La figura 4.23d muestra un circuito equivalente que emplea lógica AND-OR de dos niveles.

Podemos utilizar el multiplexor 4 a 1 de la figura 4.23 en una red de tipo árbol, como lo muestra la figura 4.24, donde usamos cuatro multiplexores para alimentar otro multiplexor 4 a 1, con lo que creamos un multiplexor 16 a 1. Podemos generar configuraciones de mayor tamaño de manera similar.

4.4.2 Multiplexores MSI estándar

74151A

El multiplexor 8 a 1 (74151A) se muestra en la figura 4.25. Para este circuito, podemos escribir la ecuación de salida como

$$\begin{aligned} Y &= \left[(\bar{C}\bar{B}\bar{A})D_0 + (\bar{C}\bar{B}A)D_1 + (\bar{C}B\bar{A})D_2 + (\bar{C}BA)D_3 \right. \\ &\quad \left. +(C\bar{B}\bar{A})D_4 + (C\bar{B}A)D_5 + (CB\bar{A})D_6 + (CBA)D_7\right]\bar{\bar{G}} \\ &= \left(\sum_{i=0}^{7} m_i D_i\right)\overline{(\bar{G})} \end{aligned} \tag{4.14}$$

El estroboscopio ($\overline{G}$) actúa como una señal de habilitación (activa baja), y hace que la salida sea 0 cuando $\overline{G} = 1$. La segunda salida W es el complemento de Y.

74150

El 74150 es el multiplexor 16 a 1 que se muestra en la figura 4.26. Éste es un dispositivo de 24 *pines* con dos líneas para la alimentación y la tierra, 16 líneas para entrada de datos (E_i), un estroboscopio ($\overline{G}$), cuatro líneas para selección de código (D, C, B, A) y una línea de salida (W). La salida de este dispositivo es

$$W = \overline{\left(\sum_{i=0}^{15} m_i E_i\right)\bar{\bar{G}}} \tag{4.15}$$

donde m_i es un mintérmino (D, C, B, A).

74153

Con frecuencia, los sistemas de cómputo requieren la multiplexión de varias fuentes de información (con varios bits cada una) en un *bus*. Para esto, se combinan dos o más multiplexores en un módulo, con una entrada de código de selección común. La figura 4.27 muestra un módulo con dos multiplexores 4 a 1, conocida como multiplexor dual (2 bits) de cuatro entradas. El comportamiento de este módulo se ilustra mediante el conmutador giratorio de la figura 4.27b, donde un conjunto de dos entradas $\{1Ci, 2Ci\}$ se conecta a las dos líneas de salida $\{1Y, 2Y\}$ cuando las líneas de selección $(BA)_2 = (i)_2$. La figura 4.27c muestra un símbolo alternativo para el multiplexor dual de cuatro entradas. Las figuras 4.27d y e muestran el diagrama lógico y el símbolo estándar IEEE del 74153, respectivamente.

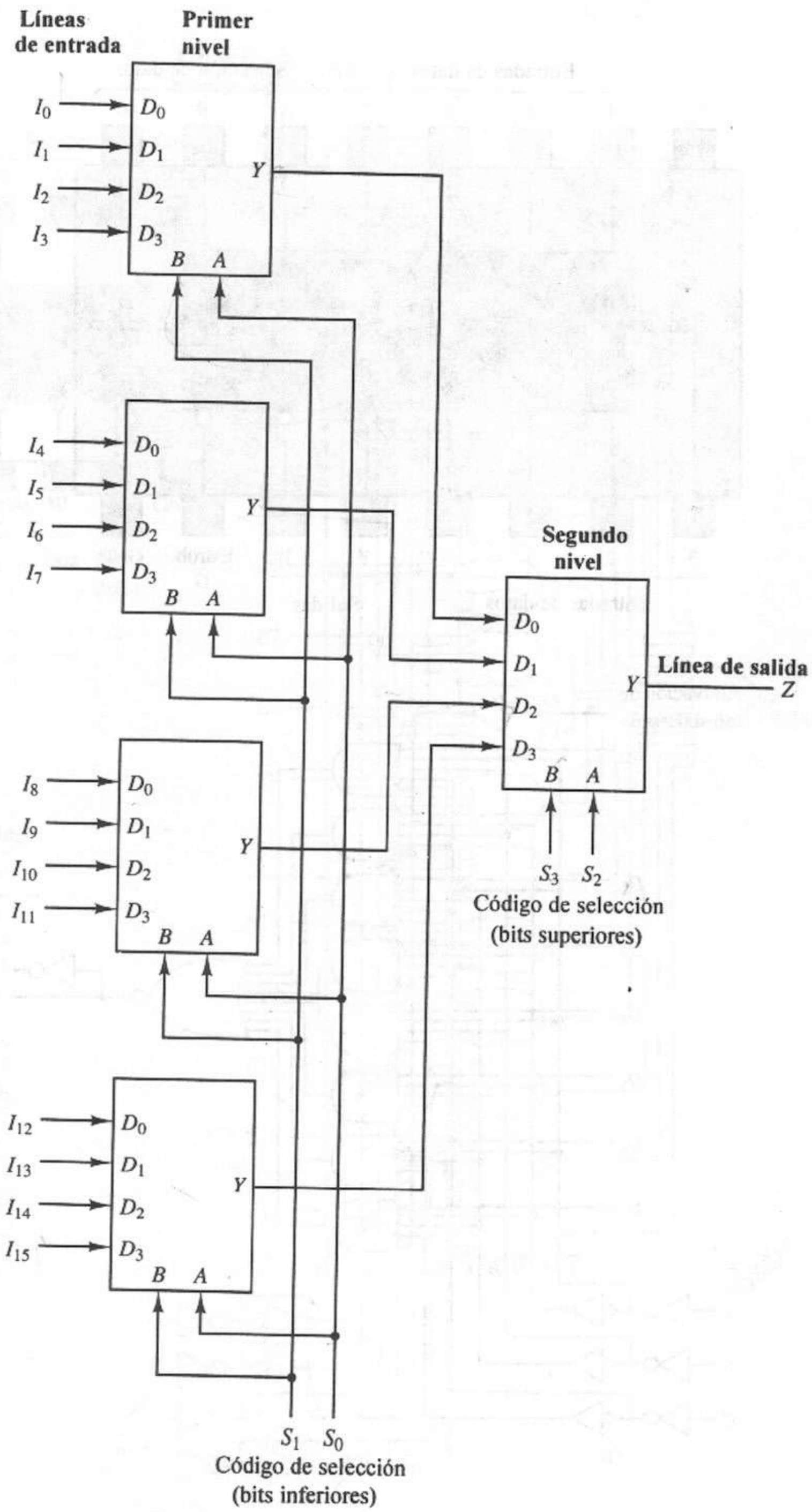
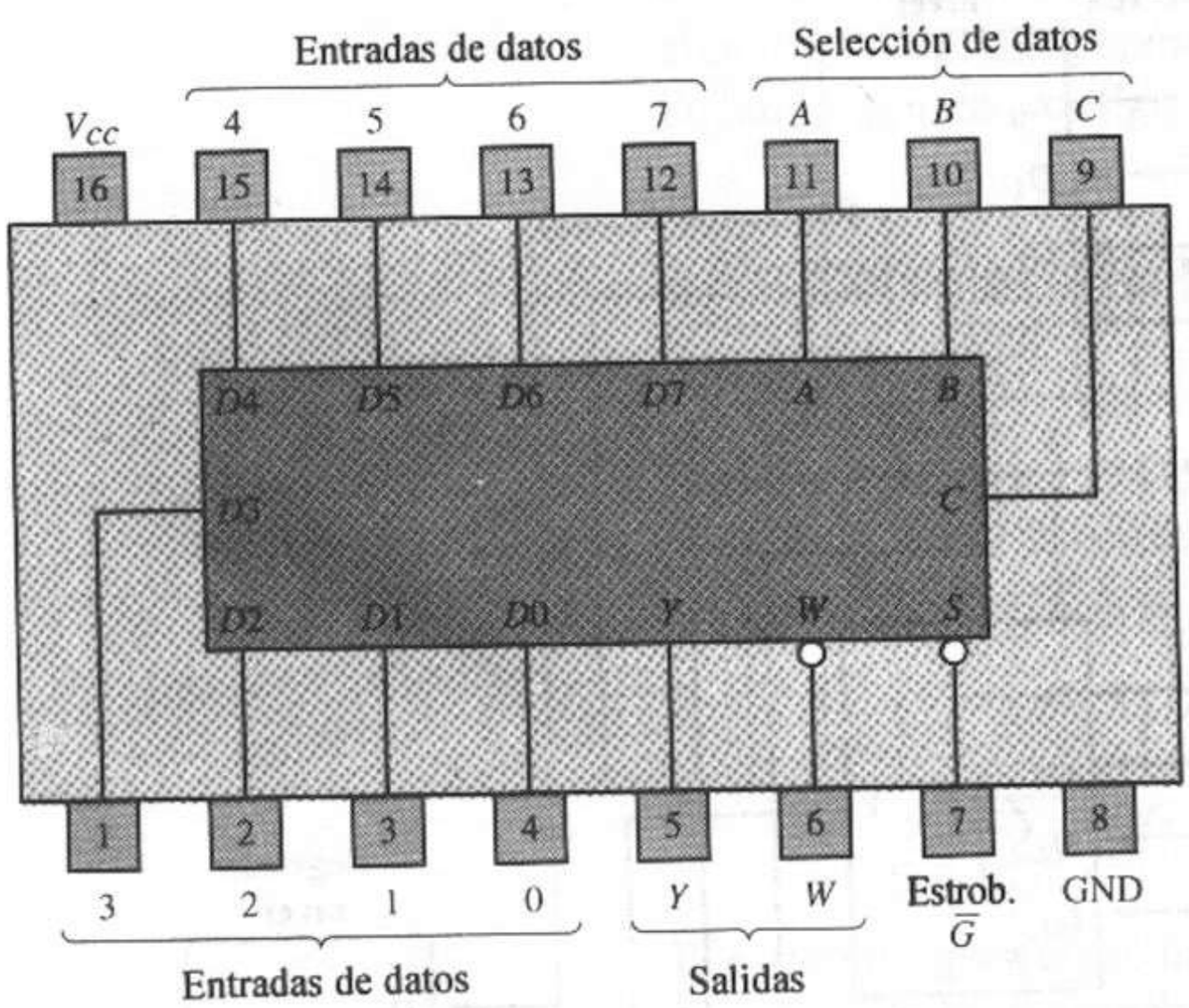


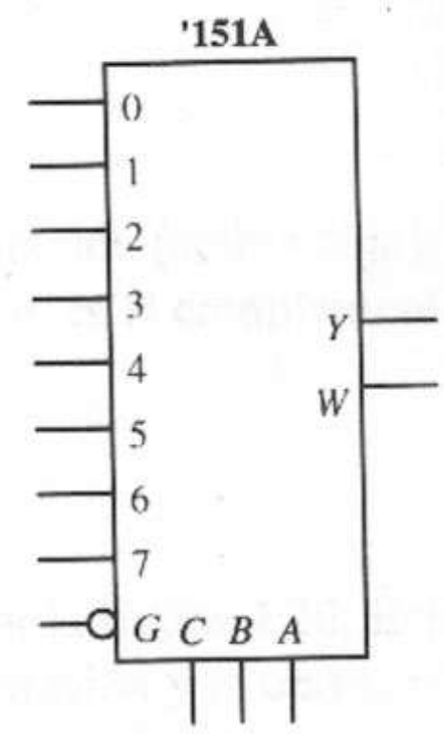
Figura 4.24 Multiplexor 16 a 1 realizado mediante una red de tipo árbol de multiplexores 4 a 1.



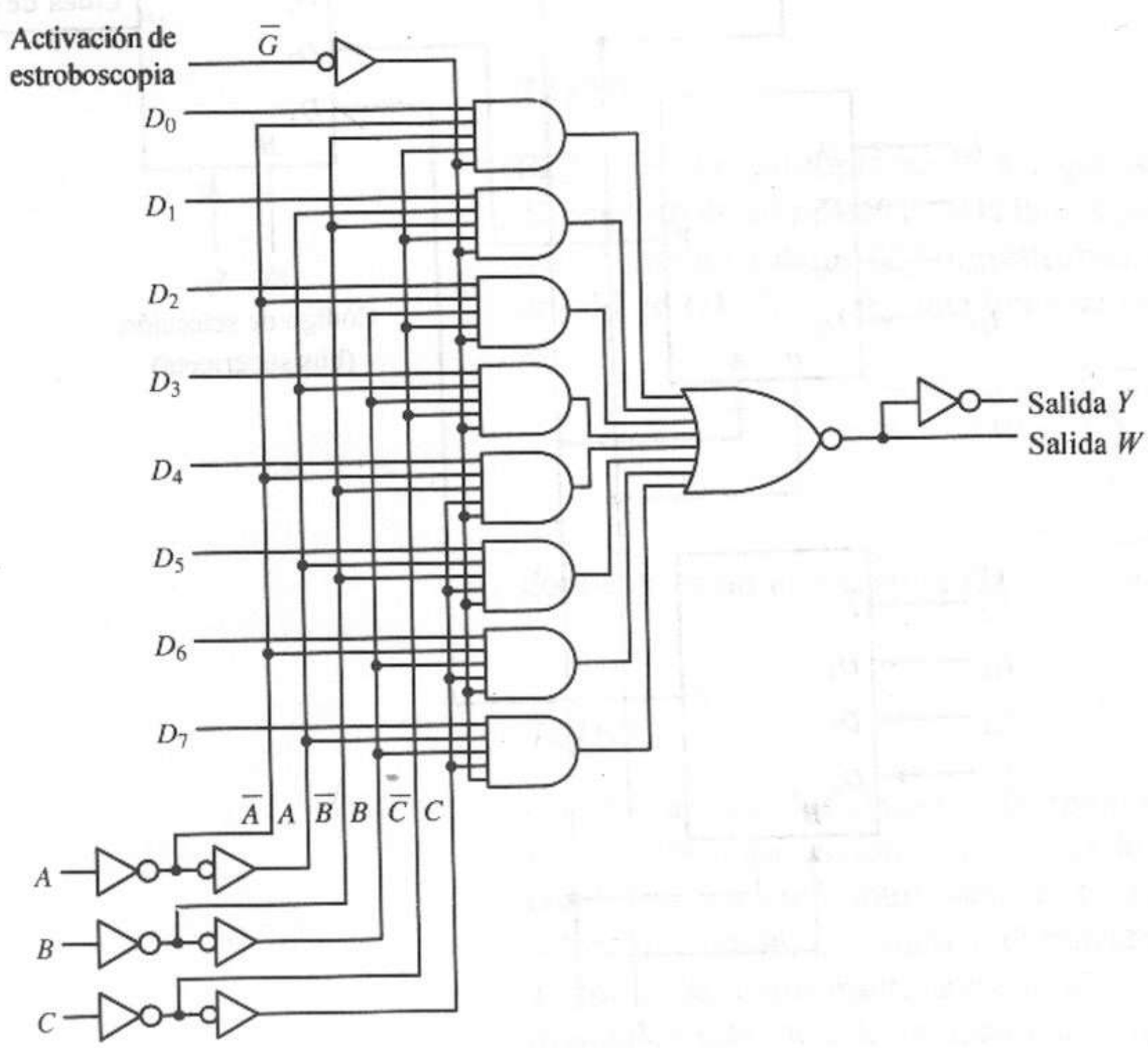
(a)

Entradas				Salidas	
Selección			Estrob.		
C	B	A	$\overline{G}$	Y	W
×	×	×	H	L	H
L	L	L	L	$D0$	$\overline{D0}$
L	L	H	L	$D1$	$\overline{D1}$
L	H	L	L	$D2$	$\overline{D2}$
L	H	H	L	$D3$	$\overline{D3}$
H	L	L	L	$D4$	$\overline{D4}$
H	L	H	L	$D5$	$\overline{D5}$
H	H	L	L	$D6$	$\overline{D6}$
H	H	H	L	$D7$	$\overline{D7}$

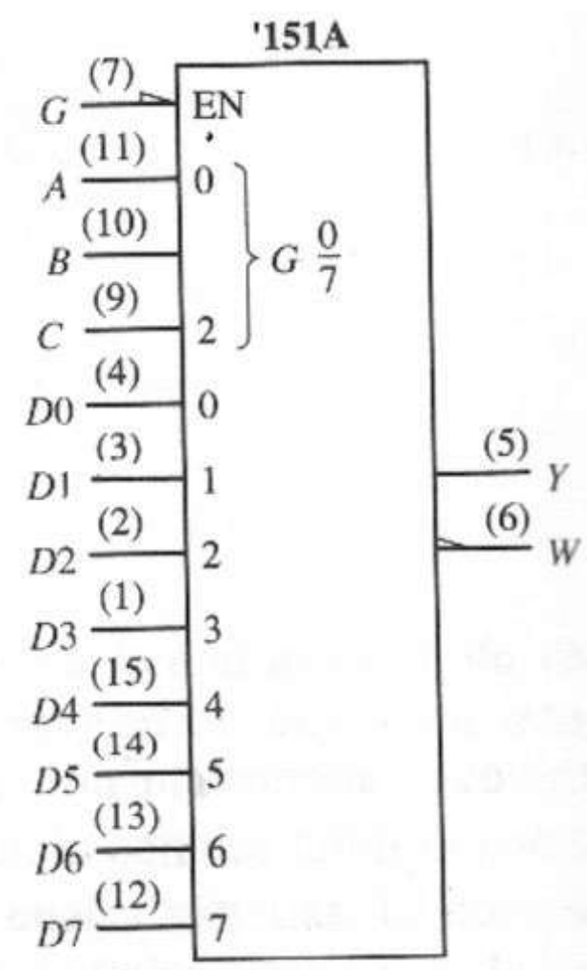
(b)



(d)

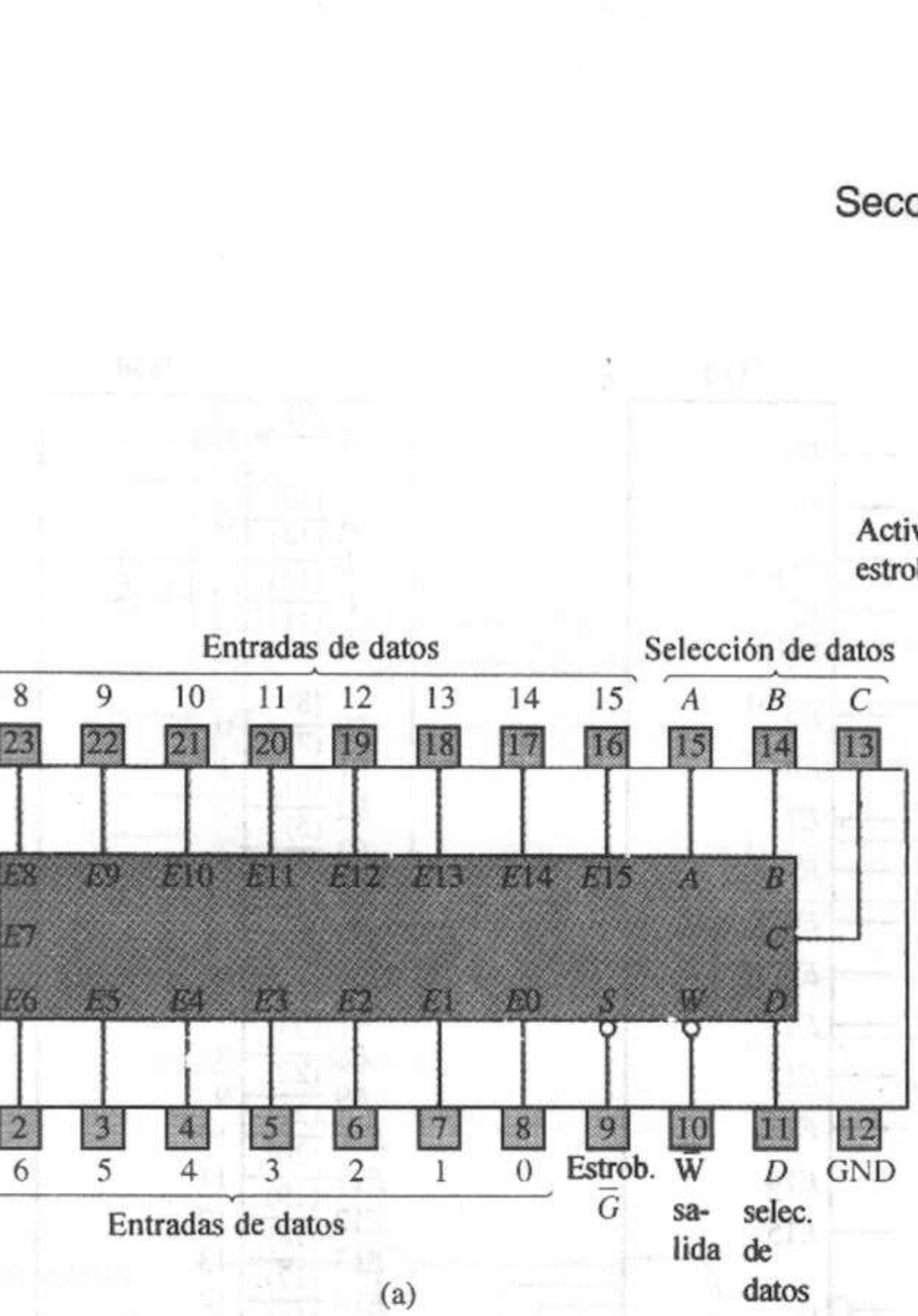


(c)



(e)

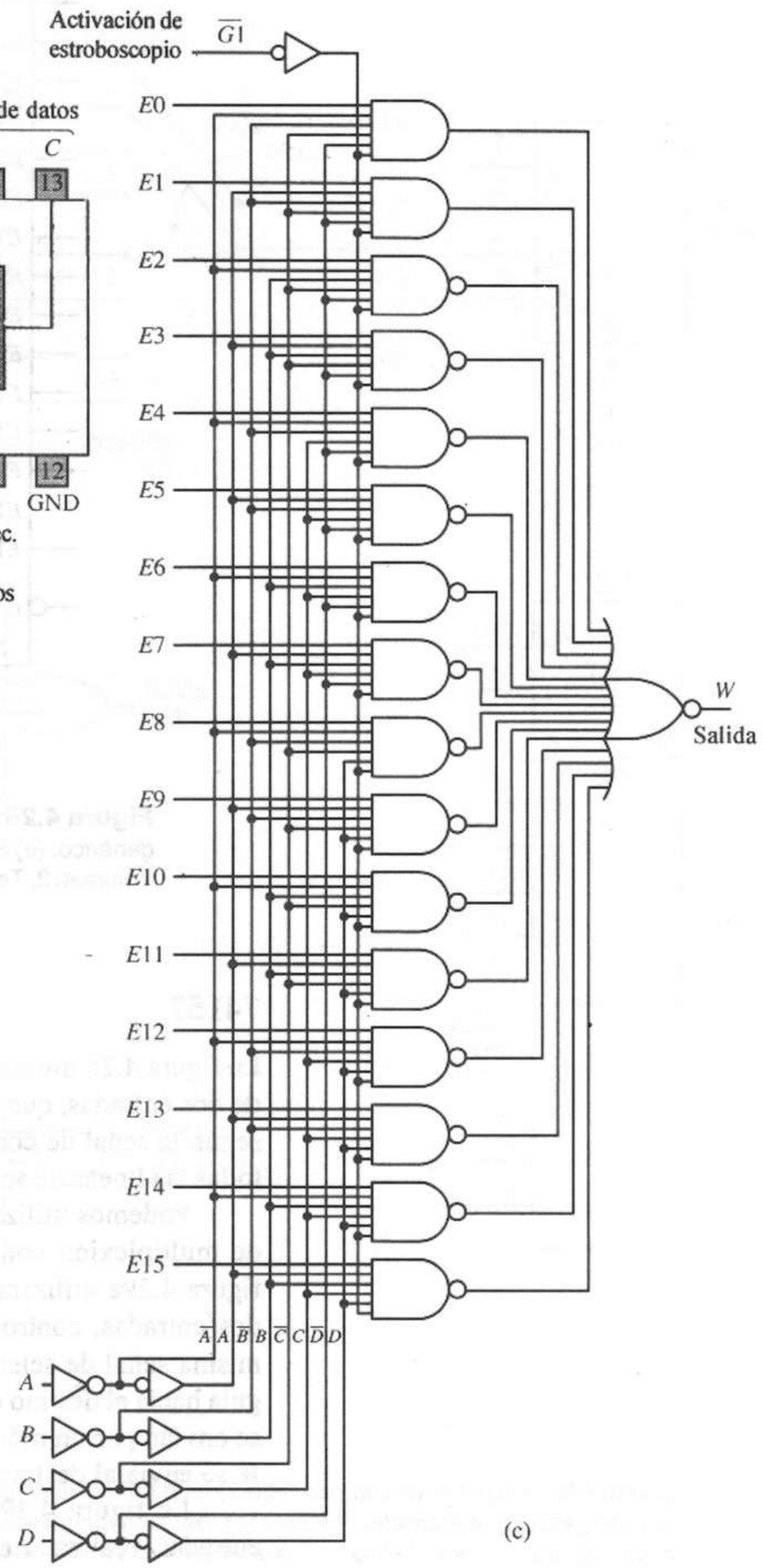
Figura 4.25 Multiplexor 74151A 8 a 1. **(a)** Configuración del paquete. **(b)** Tabla funcional. **(c)** Diagrama lógico. **(d)** Símbolo lógico genérico. **(e)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments, Inc., 1985.



(a)

Entradas					
Selección				Estrob.	Salida
D	C	B	A	$\overline{G}$	W
×	×	×	×	H	H
L	L	L	L	L	$\overline{E0}$
L	L	L	H	L	$\overline{E1}$
L	L	H	L	L	$\overline{E2}$
L	L	H	H	L	$\overline{E3}$
L	H	L	L	L	$\overline{E4}$
L	H	L	H	L	$\overline{E5}$
L	H	H	L	L	$\overline{E6}$
L	H	H	H	L	$\overline{E7}$
H	L	L	L	L	$\overline{E8}$
H	L	L	H	L	$\overline{E9}$
H	L	H	L	L	$\overline{E10}$
H	L	H	H	L	$\overline{E11}$
H	H	L	L	L	$\overline{E12}$
H	H	L	H	L	$\overline{E13}$
H	H	H	L	L	$\overline{E14}$
H	H	H	H	L	$\overline{E15}$

(b)



(c)

Figura 4.26 Multiplexor 74150 16 a 1. **(a)** Configuración del paquete. **(b)** Tabla funcional. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments, Inc., 1985.

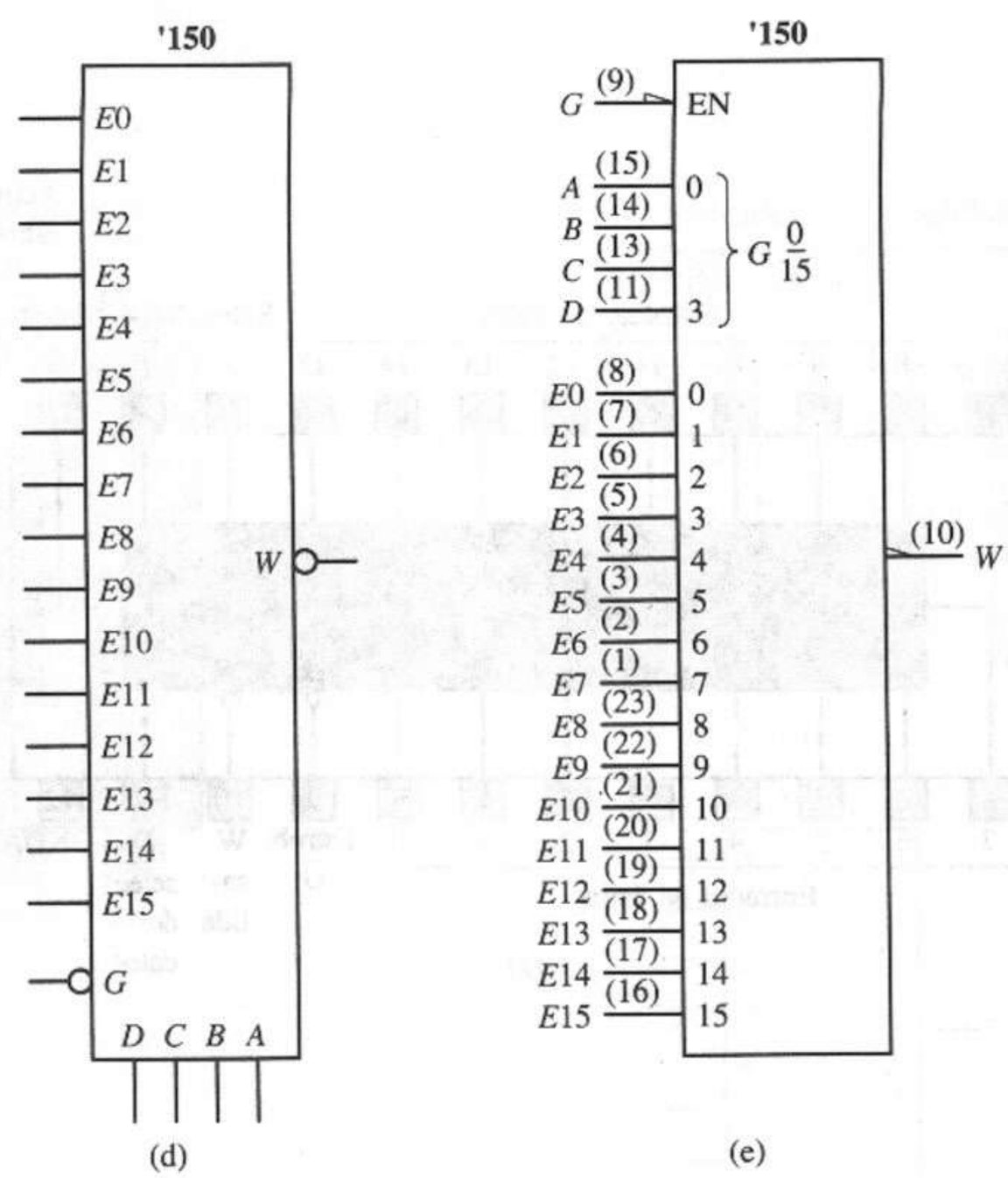


Figura 4.26 (continuación) Multiplexor 74150 16 a 1. **(d)** Símbolo lógico genérico. **(e)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments, Inc., 1985.

74157

La figura 4.28 muestra el módulo multiplexor cuádruple (de cuatro bits) 74157 de dos entradas, que conecta una de dos entradas de 4 bits a la salida de 4 bits, según la señal de control S. La señal de control $\overline{G}$ activa y desactiva la salida; todas las líneas de salida son 0 si $\overline{G}=1$.

Podemos utilizar varios módulos 74157 para crear otras configuraciones de multiplexión con diversos anchos de ruta y números de entradas. En la figura 4.29a utilizamos dos 74157 para crear un multiplexor octal (8 bits) de dos entradas, controlando la línea de selección S de ambos módulos con la misma señal de selección. En este caso, la entrada de 8 bits de la fuente X se guía hacia el destino cuando la selección es 0, de modo que los 4 bits superiores se envían por un módulo y los 4 bits inferiores se envían por el otro. La fuente W se envía al destino de la misma forma cuando la selección es 1.

La figura 4.29b muestra un multiplexor cuádruple (4 bits) de cuatro entradas, realizado con dos módulos 74157. La señal de selección $S1$ activa uno de los dos módulos y desactiva las salidas del otro, haciendo que una de las dos entradas de cada una de las compuertas OR de la figura 4.29b sea 0. La señal de selección $S0$ selecciona una de las dos entradas de 4 bits del módulo activado, enviando los 4 bits de la fuente elegida a las otras entradas de las

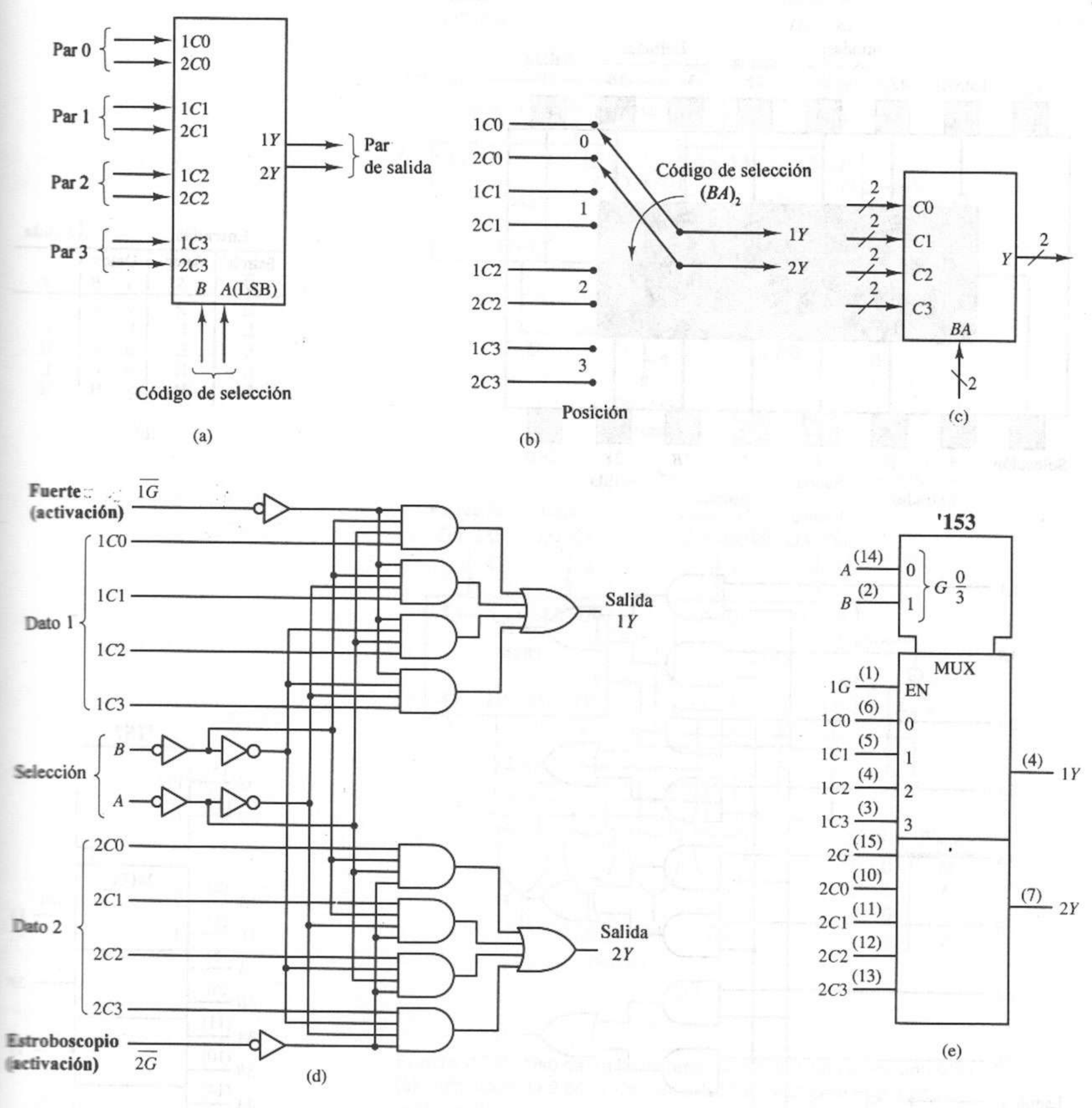
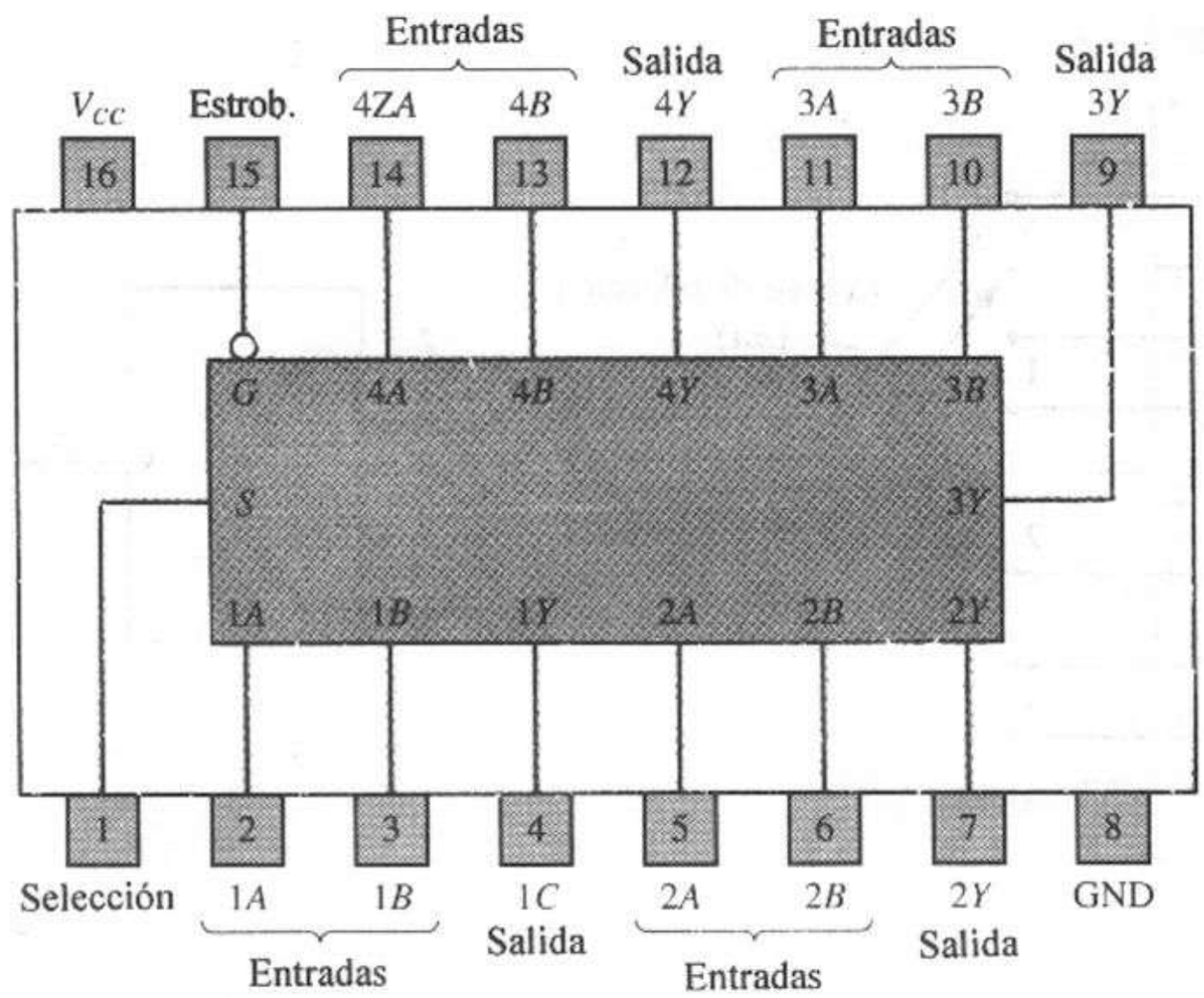


Figura 4.27 Multiplexor 74153 dual (2 bits) con cuatro entradas. **(a)** Símbolo genérico. **(b)** Conmutador equivalente. **(c)** Símbolo alternativo. **(d)** Diagrama lógico. **(e)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments, Inc., 1985.

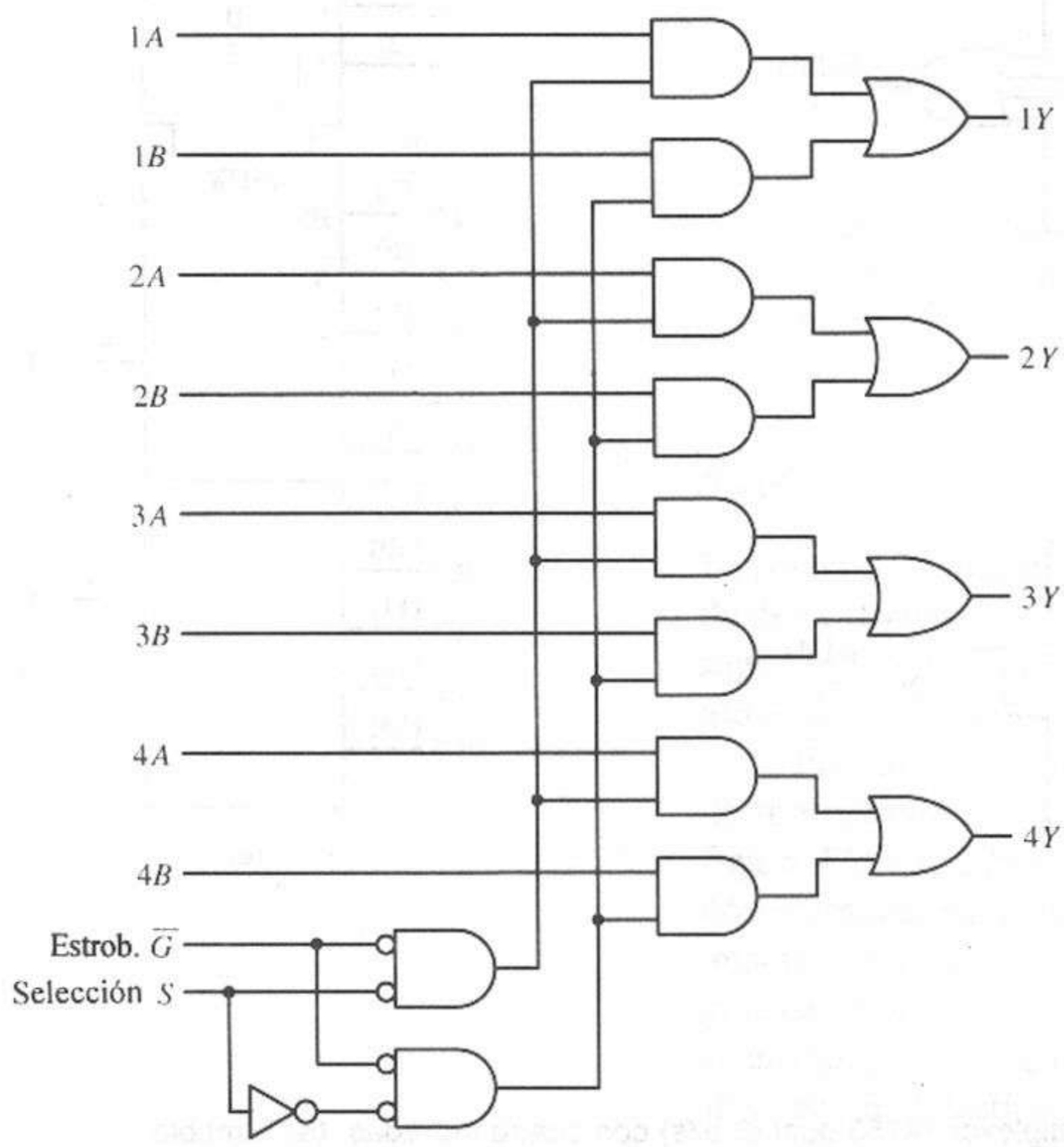
compuertas OR. En consecuencia, cada salida de compuerta OR es el bit correspondiente de la fuente elegida.



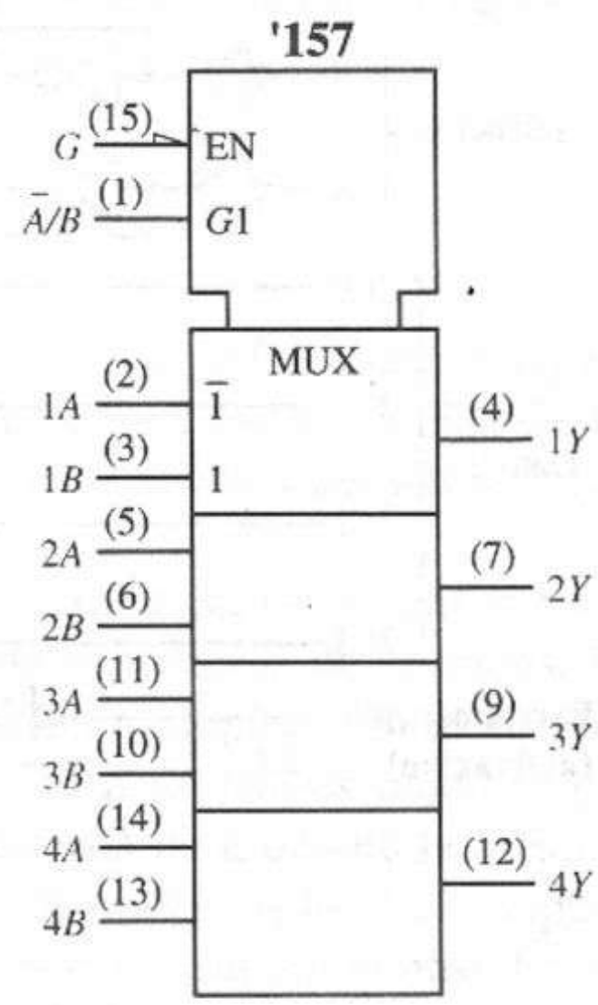
(a)

Entradas				Salida
Estrob. $\overline{G}$	Selec. S	Datos A	B	Y
H	×	×	×	L
L	L	L	×	L
L	L	H	×	H
L	H	×	L	L
L	H	×	H	H

(b)

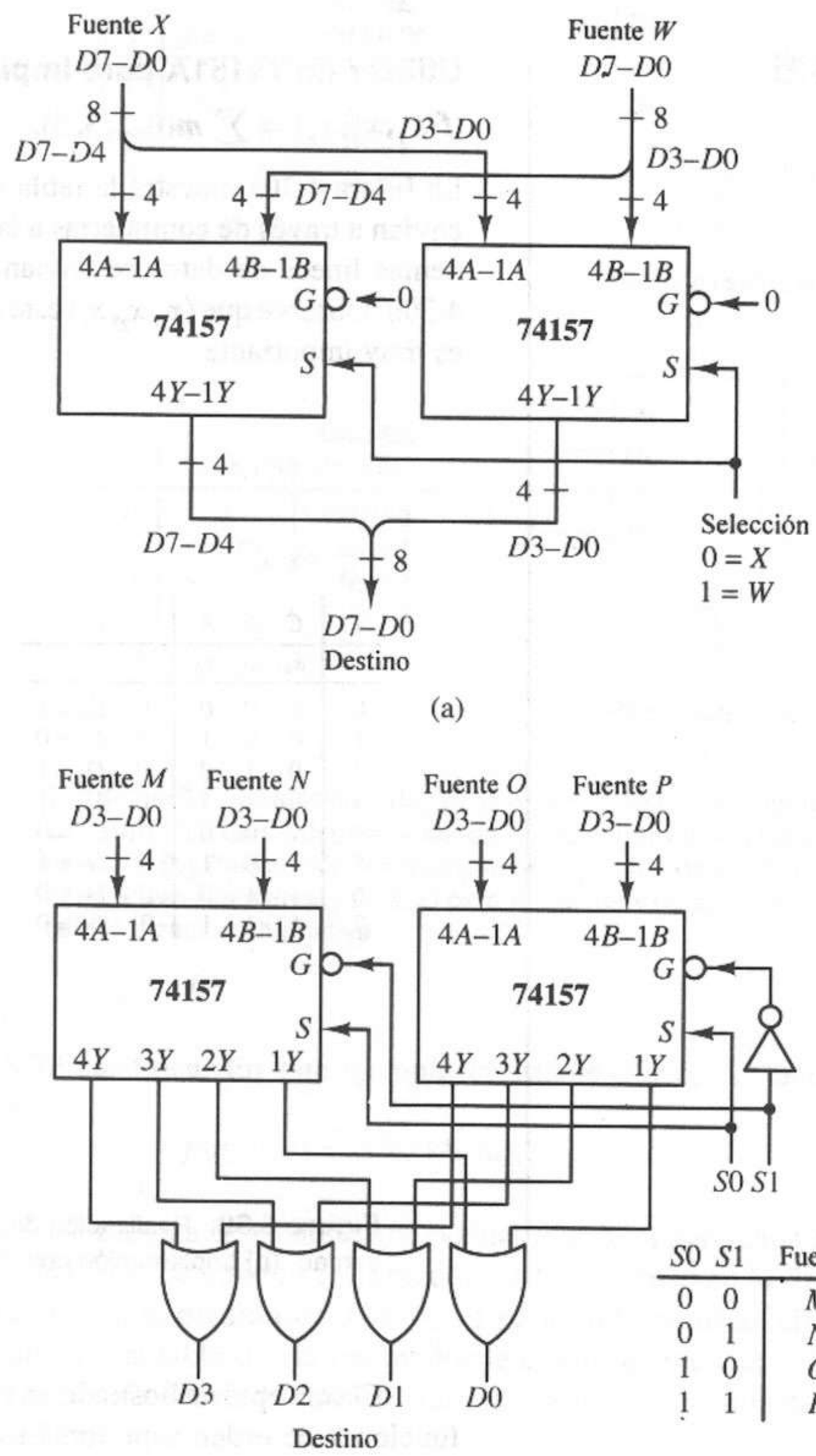


(c)



(d)

Figura 4.28 Multiplexor 74157 cuádruple 2 a 1. **(a)** Configuración de paquete. **(b)** Tabla funcional. **(c)** Diagrama lógico. **(d)** Símbolo lógico estándar IEEE. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments, Inc., 1985.



S0	S1	Fuente
0	0	M
0	1	N
1	0	O
1	1	P

Figura 4.29 Uso de multiplexores 74157 de 4 bits y dos entradas. **(a)** Multiplexor de 8 bits y dos entradas. **(b)** Multiplexor de 4 bits y cuatro entradas.

4.4.3 Aplicaciones de los multiplexores

Podemos aprovechar los multiplexores/selectores de datos presentados hasta ahora como una forma conveniente de implantar funciones de conmutación. La idea fundamental consiste en utilizar el *código de selección para generar los mintérminos* de la función, y las líneas de datos D_i, para *activar los mintérminos* presentes en un caso específico.

EJEMPLO 4.6

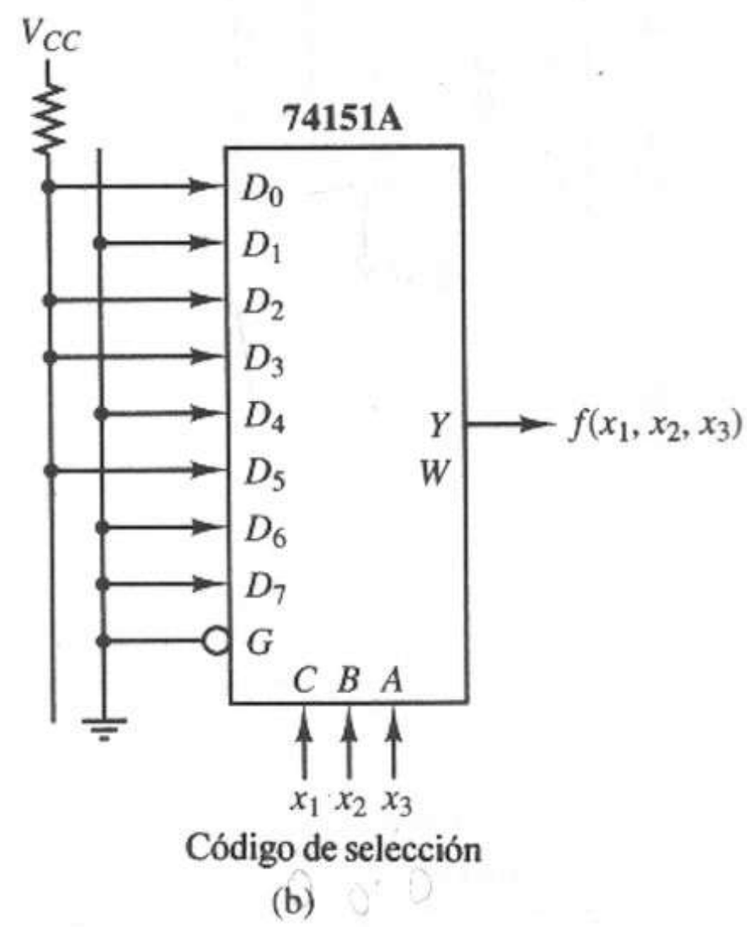
Utilizar un 74151A para implantar

$f(x_1, x_2, x_3) = \sum m(0, 2, 3, 5)$.

La figura 4.30a muestra la tabla de verdad de la función. Los mintérminos se envían a través de compuertas a la salida Y haciendo $D_0 = D_2 = D_3 = D_5 = 1$. Las demás líneas de datos se envían a una tierra, como se muestra en la figura 4.30b. Observe que (x_1, x_2, x_3) está conectado a (C, B, A). El orden de las variables es muy importante.

i	C	B	A	Y	
	x_1	x_2	x_3	f	
0	0	0	0	1	$D_0 = 1$
1	0	0	1	0	$D_1 = 0$
2	0	1	0	1	$D_2 = 1$
3	0	1	1	1	$D_3 = 1$
4	1	0	0	0	$D_4 = 0$
5	1	0	1	1	$D_5 = 1$
6	1	1	0	0	$D_6 = 0$
7	1	1	1	0	$D_7 = 0$

(a)



(b)

Figura 4.30 Realización de $f(x_1, x_2, x_3) = \Sigma m(0,2,3,5)$. **(a)** Tabla de verdad. **(b)** Implantación con 74151A.

El concepto demostrado en el ejemplo 4.6 se puede extender para implantar funciones de orden superior. Es decir, podemos utilizar un multiplexor n a 1 para implantar una función con $k + 1$ variables ($n = 2^k$) conectando k de las variables a las líneas de selección del multiplexor, donde la $(k + 1)$-ésima variable sirve (junto con la tierra y la fuente de alimentación) para establecer las líneas de entrada de los datos.

EJEMPLO 4.7

Implantar $f(a, b, c) = ab + \bar{b}c$ mediante el multiplexor 4 a 1 de la figura 4.23.

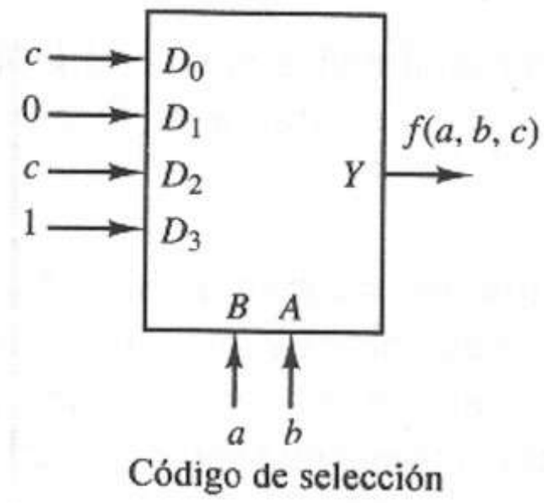
En este caso, tenemos tres variables y dos líneas de selección sobre el multiplexor. El primer paso es expresar la función en forma SOP canónica.

$$\begin{aligned} f(a, b, c) &= ab + \bar{b}c \\ &= ab\bar{c} + abc + \bar{a}\bar{b}c + a\bar{b}c \end{aligned}$$

El siguiente paso es elegir dos de las variables para conectarlas a las líneas de selección del multiplexor y sacar por factorización estos términos de la

a	b	$f(a, b, c)$	Entradas del MUX
0	0	c	$D_0 = c$
0	1	0	$D_1 = 0$
1	0	c	$D_2 = c$
1	1	1	$D_3 = 1$

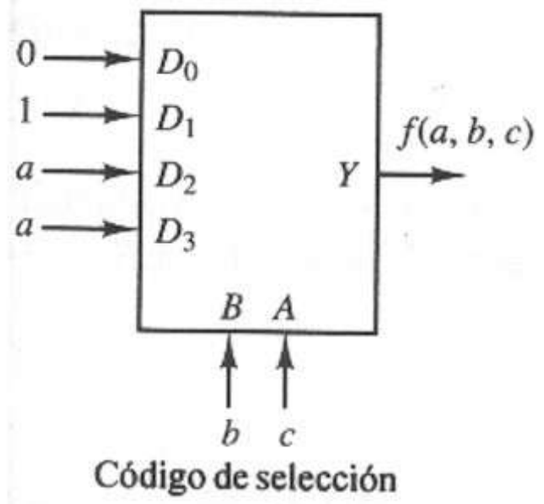
(a)



(b)

b	c	$f(a, b, c)$	Entradas de selec.
0	0	0	$D_0 = 0$
0	1	1	$D_1 = 1$
1	0	a	$D_2 = a$
1	1	a	$D_3 = a$

(c)



(d)

Figura 4.31 Realización de $f(a, b, c) = ab + \bar{b}c$ con multiplexores 4 a 1. **(a)** Tabla de verdad; evaluamos $f(a, b, c)$ para todos los valores de a y b. **(b)** Realización del multiplexor. **(c)** Tabla de verdad alternativa; evaluamos $f(a, b, c)$ para todos los valores de b y c. **(d)** Realización alternativa.

forma SOP canónica. En este ejemplo usaremos a y b. Al factorizar a y b tenemos

$$f(a, b, c) = \bar{a}\bar{b}(c) + a\bar{b}(c) + ab(\bar{c} + c)$$

Podemos utilizar esta expresión para evaluar $f(a, b, c)$ para cada combinación de a y b. El resultado aparece en forma de tabla de verdad en la figura 4.31a, la cual muestra la expresión de $f(a, b, c)$ para cada combinación de a y b. Implantamos esta tabla de verdad mediante el multiplexor de la figura 4.31b. Observe que cada fila de la tabla de verdad corresponde a una de las entradas del multiplexor.

Podemos conectar cualesquiera dos variables a las líneas de selección del multiplexor. Por ejemplo, si elegimos las variables b y c, obtenemos la tabla de verdad de la figura 4.31c. Implantamos esta tabla de verdad mediante el multiplexor de la figura 4.31d.

EJEMPLO 4.8

Implantar

$f(X_1, X_2, X_3, X_4) = \sum m(0, 1, 2, 3, 4, 9, 13, 14, 15)$

mediante un 74151A.

En este caso, la función lógica tiene cuatro variables de entrada, mientras que el multiplexor sólo tiene tres bits para el código de selección. Un método para resolver esto es factorizar los mintérminos de la función empleando el postulado 5(b).

Como la forma SOP canónica tiene muchos términos, es más sencillo omitir el paso de factorización y trabajar en forma directa con la tabla de verdad de la función, dada en la figura 4.32a.

Observe que hemos separado las variables de entrada en dos grupos, y asignamos X_1, X_2, X_3 al código de selección *C, B* y *A*. Utilizaremos la cuarta variable para entrada de datos. Los tres bits del código de selección realmente dividen la tabla de verdad en ocho partes, como lo indican las líneas horizontales. Cuando aplicamos los bits X_1, X_2 y X_3 al código de selección, éstos designan una de las ocho filas dobles de la tabla. Cada fila doble podría tener el patrón de bits

$$\left.\begin{matrix}0\\0\end{matrix}\right\}\text{ 0 lógico} \qquad \left.\begin{matrix}0\\1\end{matrix}\right\}\text{ variable } X_4$$

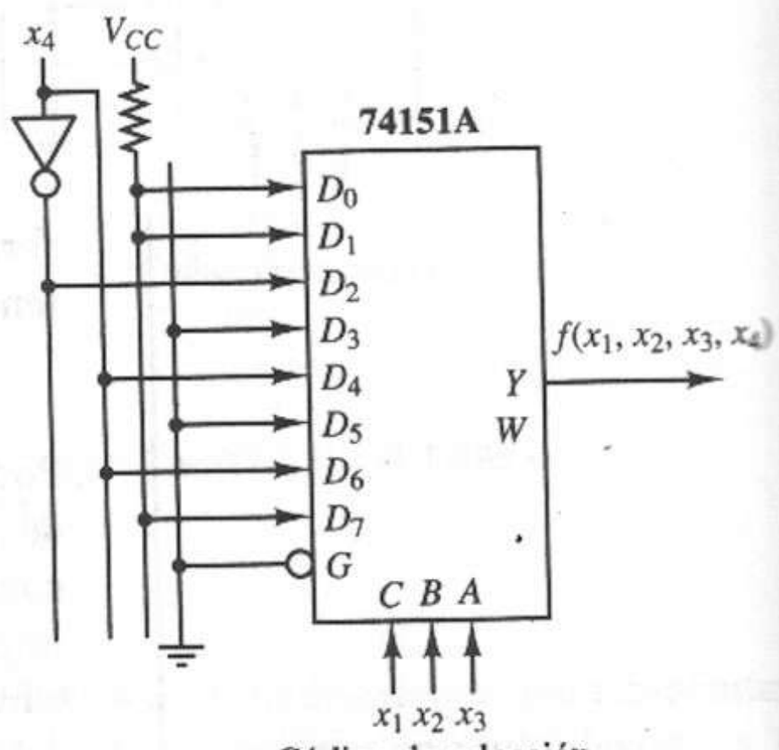
$$\left.\begin{matrix}1\\1\end{matrix}\right\}\text{ 1 lógico} \qquad \left.\begin{matrix}1\\0\end{matrix}\right\}\text{ variable } \bar{X}_4$$

Así, al elegir como entrada de datos una de estas cuatro condiciones, implantamos la función como se muestra en la figura 4.32b.

D C B A

	C	B	A				Y
i	X_1	X_2	X_3	X_4	f	f	
0	0	0	0	0	1		
	0	0	0	1	1	1	$D_0 = 1$
1	0	0	1	0	1		
	0	0	1	1	1	1	$D_1 = 1$
2	0	1	0	0	1		
	0	1	0	1	0	$\bar{X}_4$	$D_2 = \bar{X}_4$
3	0	1	1	0	0		
	0	1	1	1	0	0	$D_3 = 0$
4	1	0	0	0	0		
	1	0	0	1	1	X_4	$D_4 = X_4$
5	1	0	1	0	0		
	1	0	1	1	0	0	$D_5 = 0$
6	1	1	0	0	0		
	1	1	0	1	1	X_4	$D_6 = X_4$
7	1	1	1	0	1		
	1	1	1	1	1	1	$D_7 = 1$

(a)



(b)

Figura 4.32 Realización de $f(x_1, x_2, x_3, x_4) = \Sigma\, m(0, 1, 2, 3, 4, 9, 13, 14, 15)$ con un multiplexor 74151A. **(a)** Tabla de verdad. **(b)** Diagrama lógico.

4.5 Demultiplexores/distribuidores de datos

En la sección anterior examinamos un circuito lógico combinatorio que realiza la multiplexión de *n* líneas en una usando un código de selección para especificar la línea de entrada que debe conectarse a la línea de salida. En esta sección examinaremos el circuito inverso, un *demultiplexor* o *distribuidor de datos*.

Un demultiplexor conecta una sola línea de entrada a una de n líneas de salida, según lo determine un código de selección de s bits, donde

$$2^s \geq n \tag{4.16}$$

La figura 4.33a muestra un diagrama funcional para un demultiplexor 1 a n. Utilizamos el código de selección para generar un mintérmino de s variables; después, ese mintérmino envía el dato de entrada a través de una compuerta hacia la terminal de salida adecuada. La figura 4.33b tiene un ejemplo específico. Este distribuidor de datos 1 a 4 tiene una señal de activación (E) que controla la operación del circuito. Cuando E es 1, el circuito puede operar. Así, podemos describir el funcionamiento de este dispositivo como

$$Y_i = (m_i D)E \tag{4.17}$$

donde D es la señal de entrada que debe distribuirse a las n líneas de salida. Compare la ecuación 4.17 con la ecuación 4.6. Para el 74154, vemos que

$$Y_i = \overline{m_i(\overline{\overline{G1}} \cdot \overline{\overline{G2}})}$$

Si especificamos que

1. las entradas (D, C, B, A) del código de selección del 74154 generan los m_i en las ecuaciones 4.17 y 4.6, y
2. $G2$ en la ecuación 4.6 es el complemento de la señal de activación (E) de la ecuación 4.17,

entonces podemos utilizar el decodificador 74154 4 a 16 de la figura 4.9 como un demultiplexor 1 a 16.

Consideremos el funcionamiento del 74154 con $\overline{G2} = 0$ y un código de selección específico aplicado, como $(D, C, B, A) = (1110)$:

$$Y_i = 1 \text{ para } i \neq 14$$

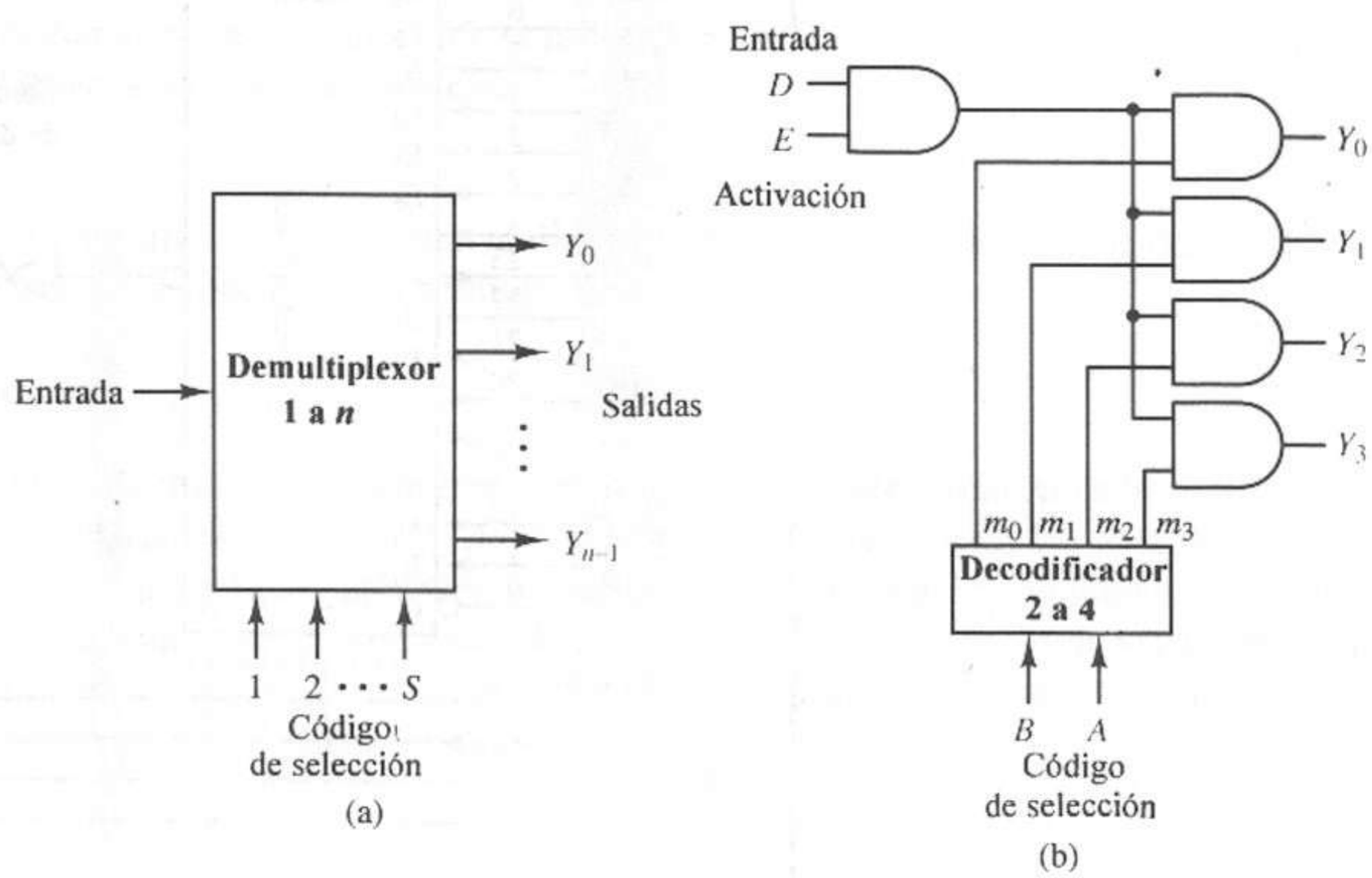


Figura 4.33 Distribuidor de datos/demultiplexor. **(a)** Diagrama funcional. **(b)** Demultiplexor 1 a 4 con activación.

y $$Y_{14} = \overline{m_{14}(\overline{\overline{G1}} \cdot \overline{\overline{G2}})}.$$

Pero $m_{14} = 1$ y $\overline{\overline{G2}} = 1$, de modo que

$$Y_{14} = \overline{\overline{\overline{G1}}} = \overline{G1}$$

Así, si hacemos $\overline{G1} = D$, el 74154 implanta la ecuación 4.17 para el demultiplexor.

EJEMPLO 4.9

Diseñar un sistema multiplexor/demultiplexor de 16 a 16 líneas mediante un 74150 y un 74154.

El objetivo es reemplazar un cable de 16 líneas por un número menor de señales para ahorrar cable. Supongamos que las 16 señales deben transportarse a 1000 pies. La figura 4.34 muestra una solución, en la que las 16 señales (X_0, X_1,..., X_{15}) se multiplexan en una línea ($\overline{Q}$) mediante el código de señal de canal (C_3, C_2, C_1, C_0). En el otro extremo, utilizamos las cinco líneas para realizar una de multiplexión de los datos, de nuevo a 16 líneas paralelas, para su procesamiento posterior. Es importante observar que la reducción de líneas de 16 a 5 se ha logrado a costa de la utilidad del sistema, ya que ahora en un instante dado sólo se puede utilizar uno de los 16 canales de señal. Es decir, hay que asignar espacios de tiempo a cada una de las 16 líneas, y hay que establecer una programación de tiempos para la línea $\overline{Q}$. En esta configuración, las 16 líneas de salida están altas cuando no están programadas para utilizar el único canal de entrada Q.

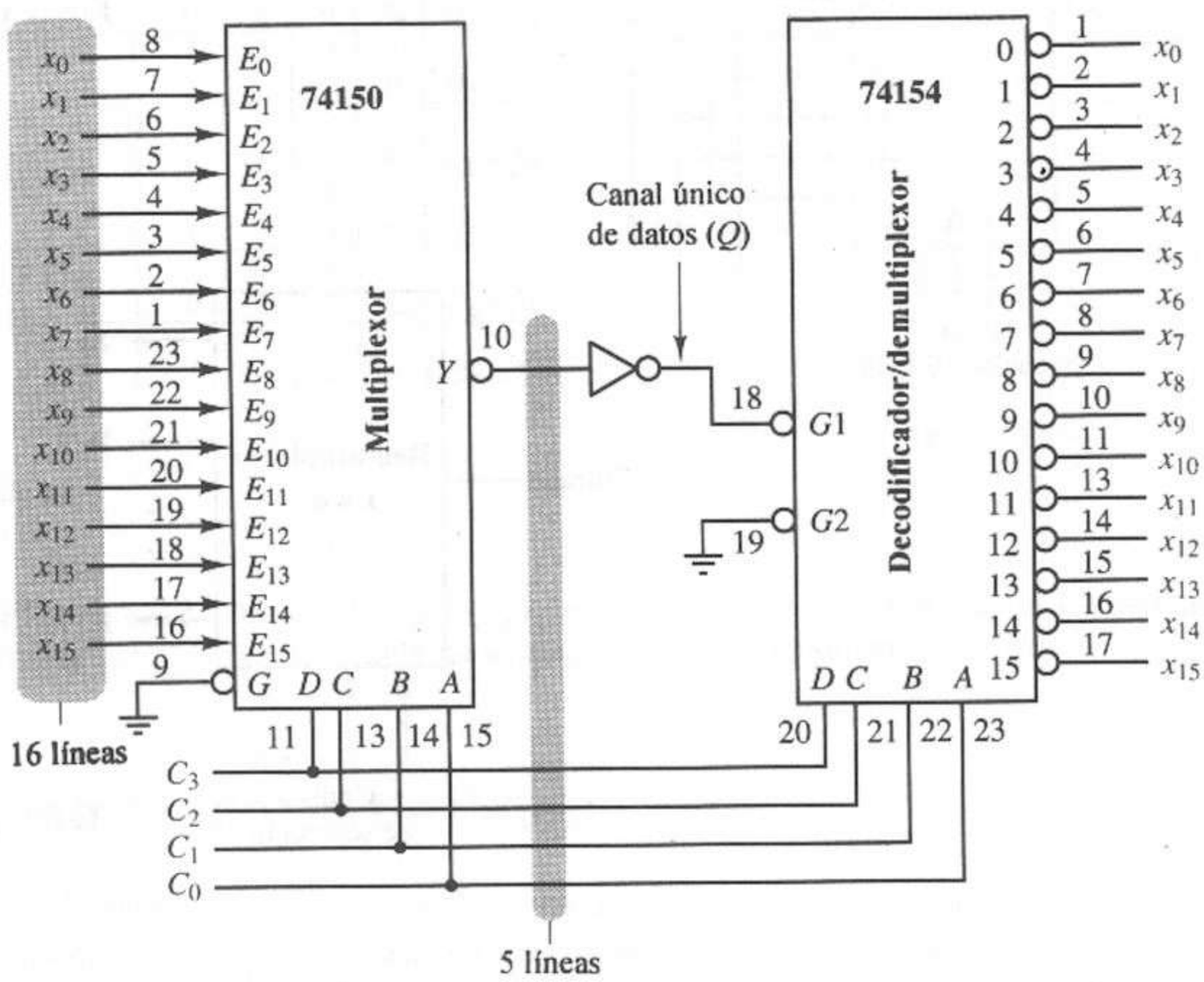


Figura 4.34 Ejemplo de distribuidor de datos/demultiplexor.

4.6 Elementos de aritmética binaria

En el capítulo 1 presentamos los fundamentos de las operaciones aritméticas en diversos sistemas numéricos. Todas las computadoras digitales tienen circuitos lógicos que implantan ciertas operaciones aritméticas en los sistemas numéricos particulares elegidos para su uso en tales computadoras. El sistema numérico de uso más frecuente para la representación de los enteros es el sistema numérico con complemento a dos, pues éste facilita la representación de valores positivos y negativos, así como la implantación de circuitos de suma y resta.

En las siguientes secciones examinaremos el diseño de los circuitos para la suma y la resta binaria que podemos utilizar para el manejo de números en forma de complemento a dos. Existen muchos otros módulos aritméticos para realizar las operaciones aritméticas binarias, incluidas la multiplicación, la división y otras. El lector debe consultar [1] si desea mayor información.

4.6.1 Circuitos sumadores binarios básicos

En muchas aplicaciones lógicas de las computadoras es necesario sumar números binarios. En el capítulo 1 mostramos que la suma de números binarios en el sistema numérico de complemento a dos basta para desarrollar las operaciones normales de suma y resta de la computadora digital. Por supuesto, los circuitos sumadores deben estar acompañados por la red de complementos y los registros aritméticos adecuados. Aquí diseñaremos varios circuitos sumadores seriales y paralelos.

Medio-sumador

Un medio-sumador (HA) es una red lógica combinatoria de varias salidas que suma 2 bits de datos binarios, produciendo señales de salida de bit de suma y bit de acarreo. Ver la figura 4.35a. Los bits de entrada x_i y y_i se suman de manera matemática en binario, como se muestra en la tabla de verdad de la figura 4.35b. Observamos en esta tabla que

$$s_i = x_i \oplus y_i \qquad (4.18)$$
$$c_i = x_i y_i$$

La figura 4.35c muestra una realización con compuertas NAND de dos entradas para estas funciones de conmutación.

Sumador completo

Al realizar la suma binaria, mostramos en el capítulo 1 que en cada posición de bit estaremos, en general, sumando dos bits de datos y un bit de acarreo. Por tanto, un sumador completo (FA) es una red lógica combinatoria de varias salidas que suma 3 bits binarios. Ver la figura 4.35d. La tabla de verdad para el sumador completo aparece en la figura 4.35e. Con base en esta tabla de verdad o un mapa K, podemos ver que

$$s_i = x_i \oplus y_i \oplus c_{i-1} \qquad (4.19)$$
$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$$

La figura 4.35f muestra una implantación NAND del sumador completo. Esta implantación utiliza nueve compuertas y dos niveles de lógica. La figura 4.35g muestra un circuito OR-exclusivo para la misma salida s_i.

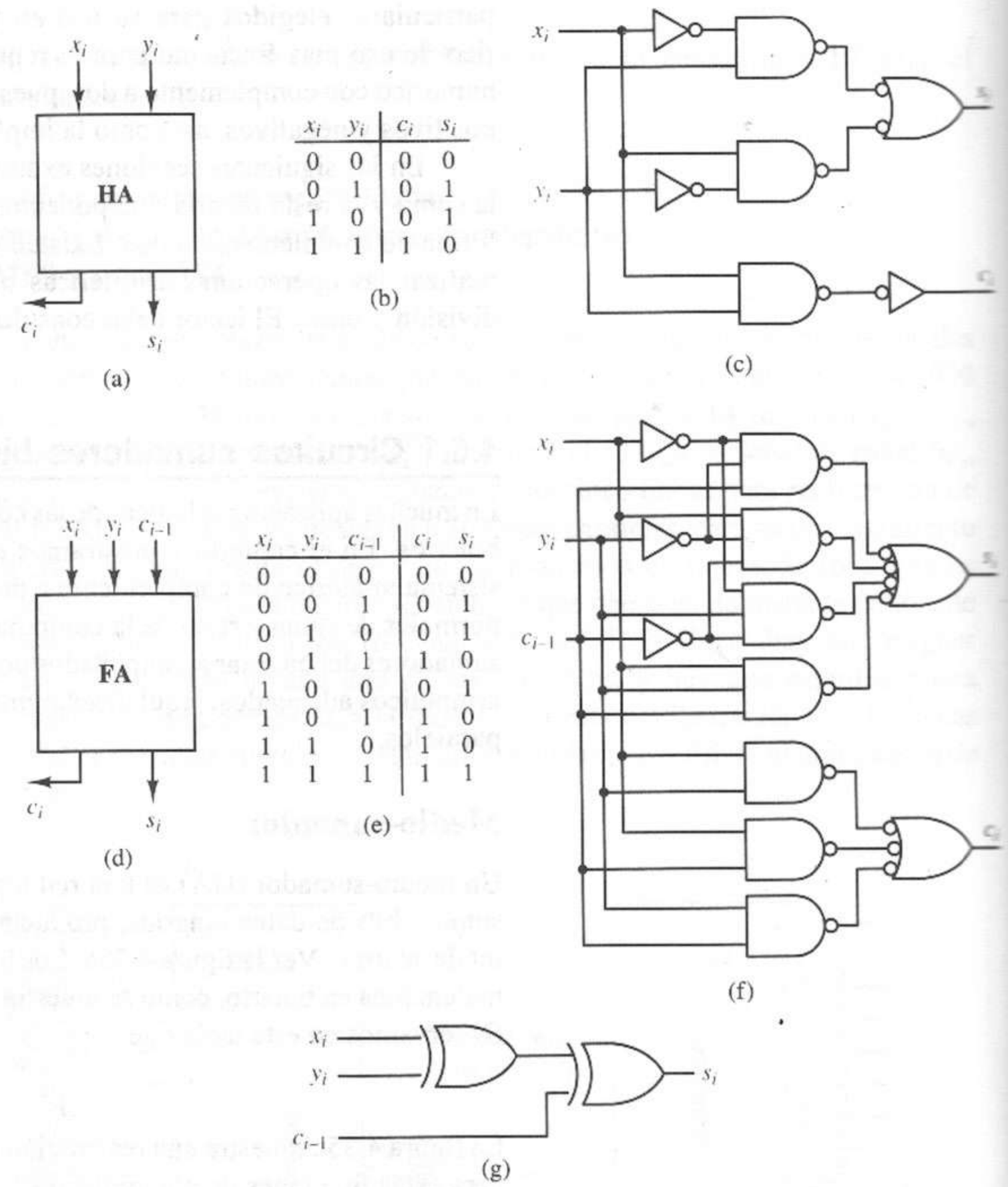


Figura 4.35 Circuitos binarios medio-sumador y sumador completo. **(a)** Medio-sumador (HA). **(b)** Tabla de verdad de HA. **(c)** Circuito HA con compuertas NAND. **(d)** Sumador completo (FA). **(e)** Tabla de verdad de FA. **(f)** Circuito FA con compuertas NAND. **(g)** Realización de la salida s_i de FA con compuertas XOR.

El diseñador de lógica digital utiliza el sumador completo como un módulo para crear circuitos mayores, y emplea el símbolo lógico de la figura 4.35d. Ahora mostraremos dos aplicaciones de este módulo.

Unidad sumadora seudoparalela

La figura 4.36 muestra una unidad sumadora que utiliza n-1 sumadores completos y un medio-sumador. Esta configuración tiene un circuito sumador

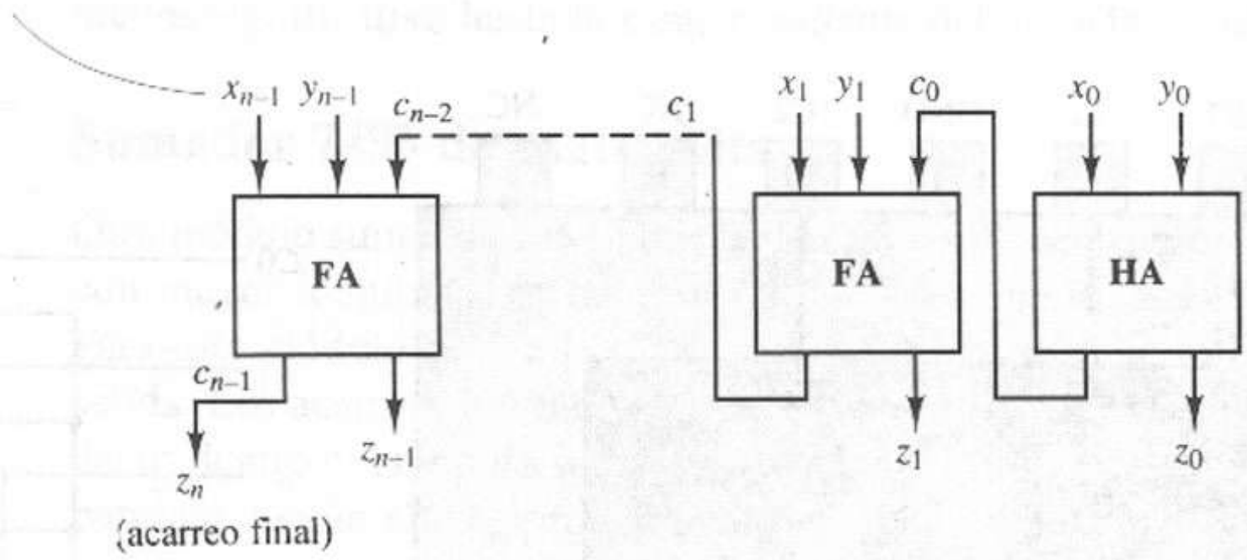


Figura 4.36 Sumador seudoparalelo (acarreo en cascada).

para cada posición de bit de las dos palabras de datos de entrada. La operación por realizar es

$$\begin{array}{rl} & X \\ + & Y \\ \hline & Z \end{array} \qquad \begin{array}{r} (x_{n-1}x_{n-2}\cdots x_1x_0)_2 \\ +(y_{n-1}y_{n-2}\cdots y_1y_0)_2 \\ \hline (z_nz_{n-1}z_{n-2}\cdots z_1z_0)_2 \end{array}$$

Esta configuración es *seudoparalela*, pues el acarreo debe propagarse a través de toda la unidad sumadora. Esta configuración también se conoce como *sumador con acarreo en cascada.* En general, la ruta de propagación en el peor de los casos es a través de un medio-sumador y $n-1$ sumadores completos desde las entradas x_0 y y_0 hasta las posiciones de suma final y de acarreo z_{n-1} y z_n.

4.6.2 Módulos de sumador binario MSI

Sumador 7482 de dos bits

El 7482 es el módulo sumador seudoparalelo de 2 bits de la figura 4.37. La señal C_0 es un acarreo de entrada y C_2 es un acarreo de salida. La señal interna $\overline{C}_1$ es el acarreo de FA_1 a FA_2. Al examinar el diagrama lógico, podemos escribir

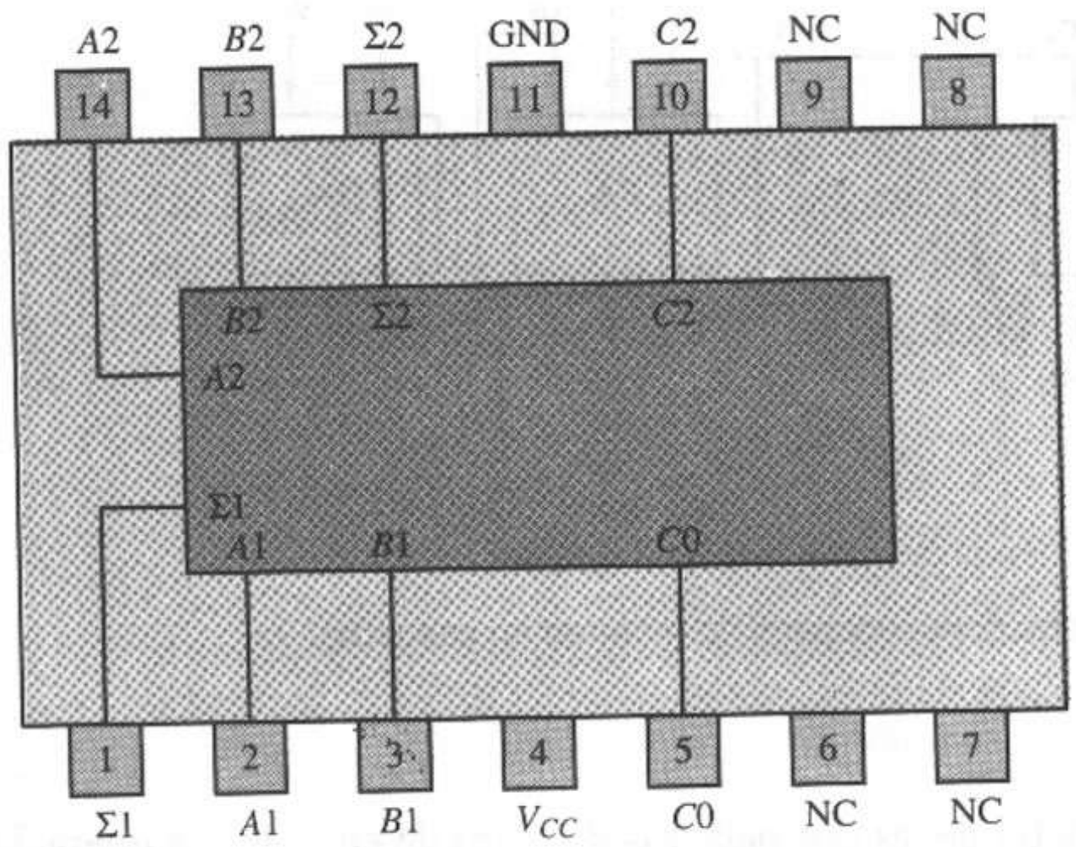
$$C_1 = C_0 \cdot A_1 + C_0 \cdot B_1 + A_1 \cdot B_1 \tag{4.20}$$

$$\begin{aligned} \Sigma_1 &= C_0 \cdot \bar{C}_1 + A_1 \cdot \bar{C}_1 + B_1 \cdot \bar{C}_1 + A_1 \cdot B_1 \cdot C_0 \\ &= \bar{C}_1(C_0 + A_1 + B_1) + A_1 \cdot B_1 \cdot C_0 \\ &= (\bar{C}_0 + \bar{A}_1)(\bar{C}_0 + \bar{B}_1)(\bar{A}_1 + \bar{B}_1)(C_0 + A_1 + B_1) + A_1 \cdot B_1 \cdot C_0 \\ &= (\bar{C}_0 + \bar{A}_1 \cdot \bar{B}_1)(\bar{A}_1 + \bar{B}_1)(C_0 + A_1 + B_1) + A_1 \cdot B_1 \cdot C_0 \qquad (4.21) \\ &= [\bar{C}_0(A_1 + B_1) + C_0 \cdot \bar{A}_1 \cdot \bar{B}_1](\bar{A}_1 + \bar{B}_1) + A_1 \cdot B_1 \cdot C_0 \\ &= \bar{C}_0 \cdot \bar{A}_1 \cdot B_1 + \bar{C}_0 \cdot A_1 \cdot \bar{B}_1 + C_0 \cdot \bar{A}_1 \cdot \bar{B}_1 + A_1 \cdot B_1 \cdot C_0 \\ &= C_0 \oplus A_1 \oplus B_1 \end{aligned}$$

De manera similar,

$$\begin{aligned} C_2 &= C_1 \cdot A_2 + C_1 \cdot B_2 + A_2 \cdot B_2 \qquad (4.22) \\ \Sigma_2 &= C_1 \oplus A_2 \oplus B_2 \end{aligned}$$

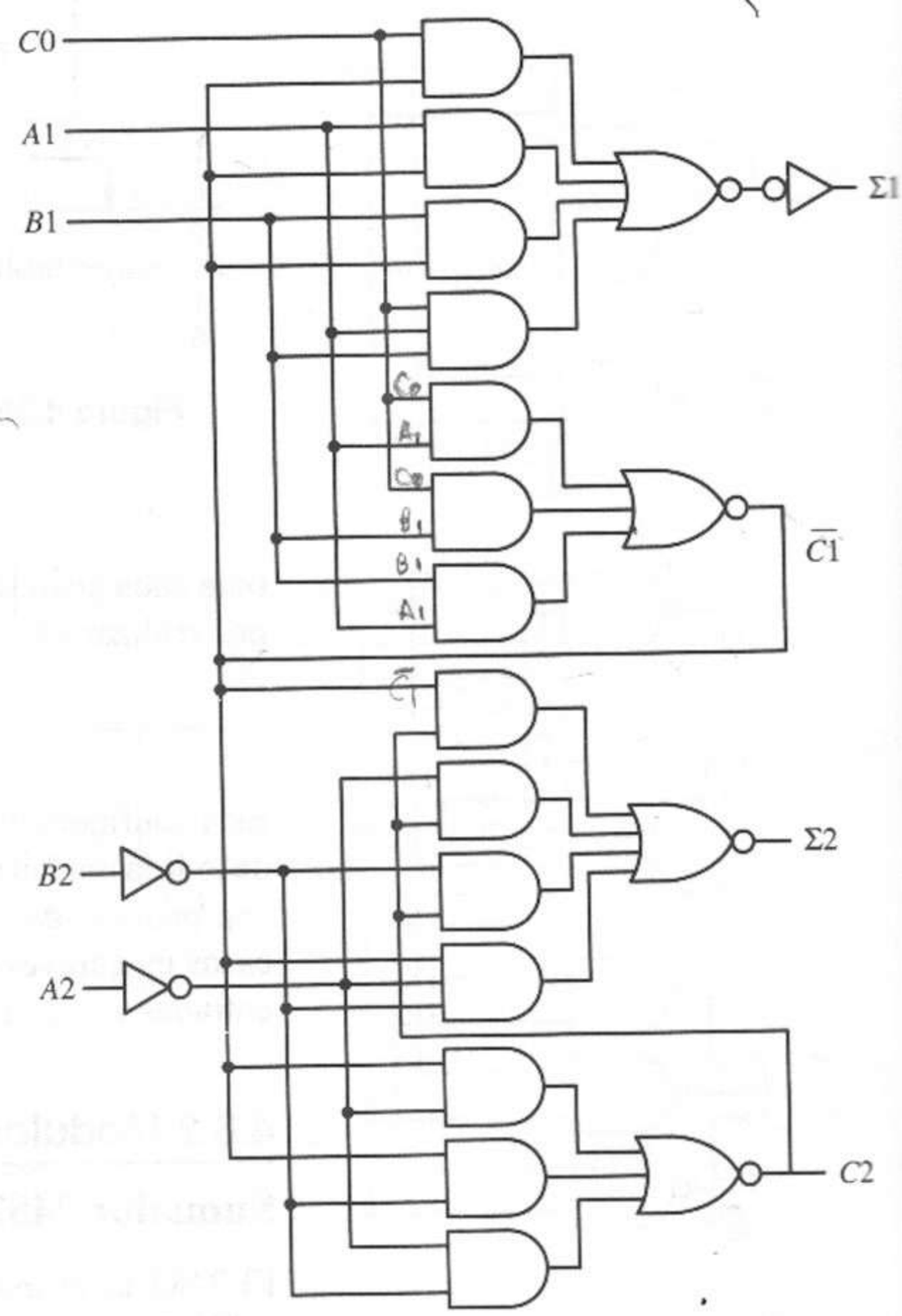
En cada sumador completo, el acarreo se genera mediante dos niveles de lógica. Después, el acarreo se combina con las señales de entrada para generar la suma con otros dos niveles de lógica. La propagación del acarreo sólo requiere dos



(a)

Entradas				Salidas					
				Cuando $C0 = L$			Cuando $C0 = H$		
$A2$	$A1$	$B2$	$B1$	$C2$	$\Sigma2$	$\Sigma1$	$C2$	$\Sigma2$	$\Sigma1$
L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	H	L	H	L
L	L	H	L	L	H	L	L	H	H
L	L	H	H	L	H	H	H	L	L
L	H	L	L	L	L	H	L	H	L
L	H	L	H	L	H	L	L	H	H
L	H	H	L	L	H	H	H	L	L
L	H	H	H	H	L	L	H	L	H
H	L	L	L	L	H	L	L	H	H
H	L	L	H	L	H	H	H	L	L
H	L	H	L	H	L	L	H	L	H
H	L	H	H	H	L	H	H	H	L
H	H	L	L	L	H	H	H	L	L
H	H	L	H	H	L	L	H	L	H
H	H	H	L	H	L	H	H	H	L
H	H	H	H	H	H	L	H	H	H

(b)



(c)

Figura 4.37 El módulo sumador seudoparalelo 7482. **(a)** Configuración de *pines* del paquete. **(b)** Tabla funcional. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

niveles de lógica en cada sumador completo. Por tanto, podemos decir, en general, que el retardo en el peor de los casos, $t_{retardo}$, para un sumador seudoparalelo construido mediante estos módulos es

$$t_{retardo} = (2n + 2)t_{compuerta} \tag{4.23}$$

donde n es la longitud de palabra de la unidad sumadora, $t_{retardo}$ se mide en términos del número total de niveles de lógica por los que deben propagarse los acarreos, de modo que cada nivel retarda la señal en una cantidad de tiempo

$t_{compuerta}$. La ruta en el peor de los casos va desde el acarreo de entrada del bit menos significativo hasta la suma resultante del bit más significativo.

Sumador 7483 de cuatro bits

Otro módulo sumador que se puede utilizar para construir unidades sumadoras con mayor longitud de palabra es el 7483, que se muestra en la figura 4.38. Para este módulo de 4 bits, C_0 es el acarreo de entrada y C_4 es el acarreo de salida. Los acarreos internos (C_1, C_2, C_3) no se propagan en forma explícita; es decir, dentro del módulo sumador los bits de salida (Σ_1, Σ_2, Σ_3, Σ_4) se forman en paralelo. Según el diagrama lógico, podemos escribir

$$\begin{aligned} P_i &= (\overline{B_i \cdot A_i})(A_i + B_i) \\ &= (\bar{A}_i + \bar{B}_i)(A_i + B_i) \\ &= A_i \oplus B_i \end{aligned} \tag{4.24}$$

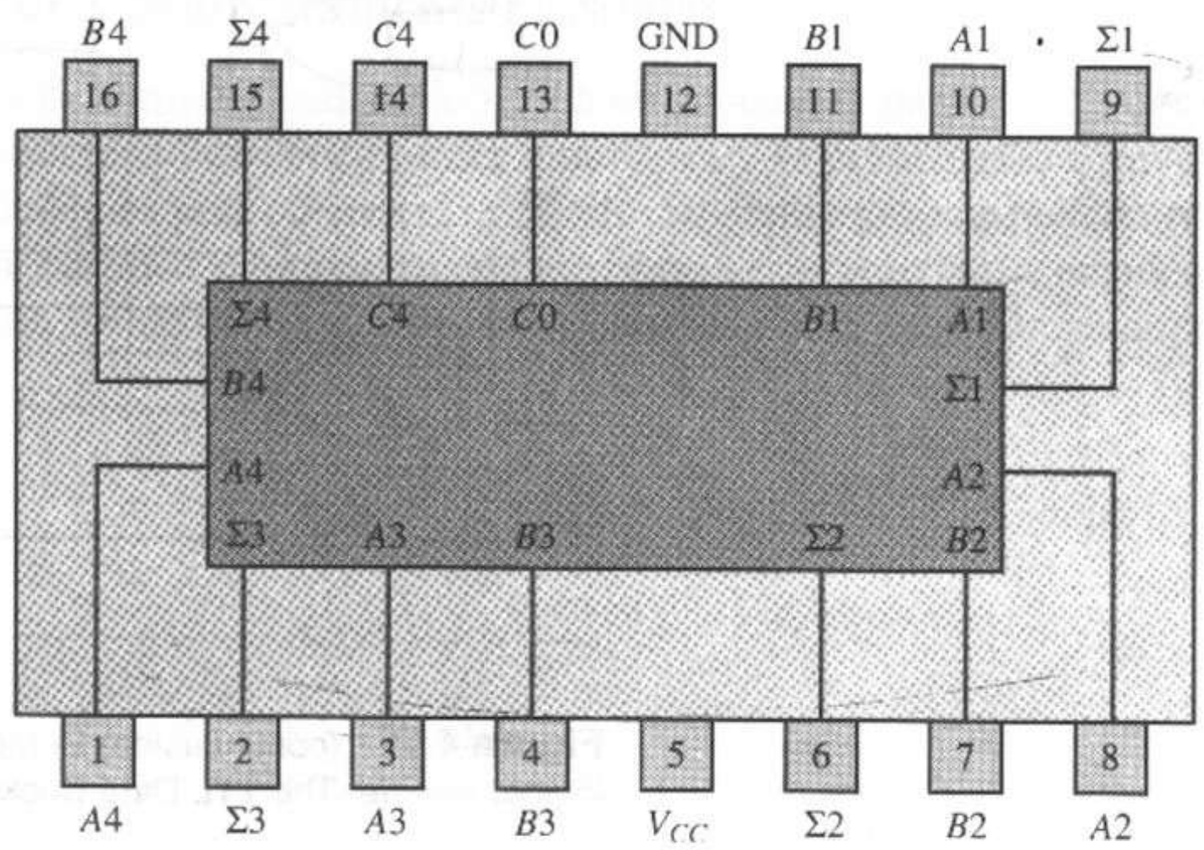
y

$$\begin{aligned} \Sigma_i &= P_i \oplus C_{i-1} \\ &= A_i \oplus B_i \oplus C_{i-1} \end{aligned} \tag{4.25}$$

$$\begin{aligned} C_1 &= \overline{(\bar{C}_0 \cdot \overline{A_1 \cdot B_1}) + (\overline{A_1 + B_1})} \\ &= \overline{(\bar{C}_0 \cdot \overline{A_1 \cdot B_1})} \cdot (A_1 + B_1) \\ &= (C_0 + (A_1 \cdot B_1)) \cdot (A_1 + B_1) \\ &= C_0 \cdot A_1 + C_0 \cdot B_1 + A_1 \cdot B_1 \end{aligned} \tag{4.26}$$

De manera análoga, podemos ver que

$$C_i = C_{i-1} \cdot A_i + C_{i-1} \cdot B_i + A_i \cdot B_i \tag{4.27}$$



(a)

Figura 4.38 El módulo sumador de cuatro bits 7483. **(a)** Configuración de *pines* del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

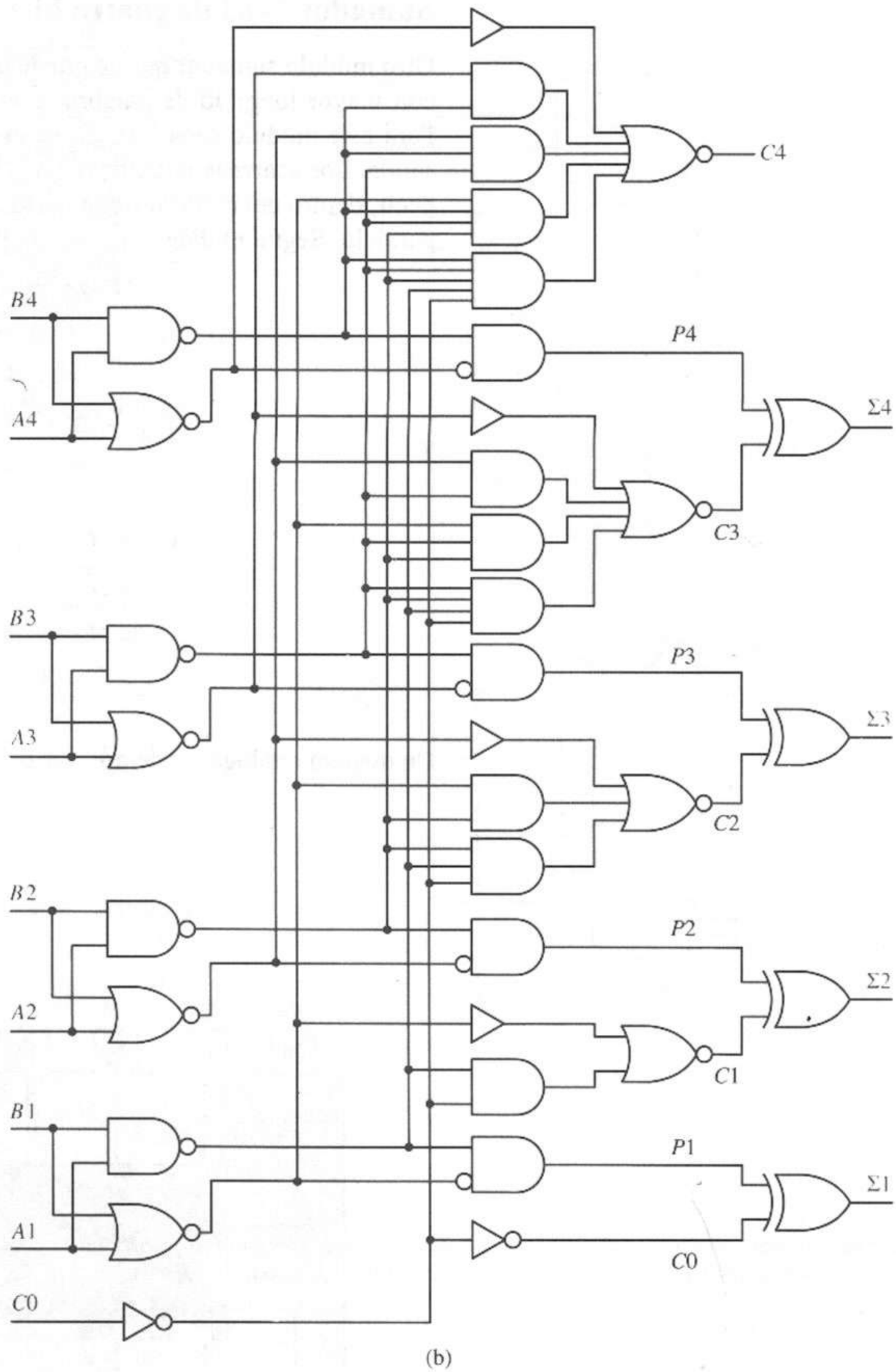


Figura 4.38 (continuación) El módulo sumador de cuatro bits 7483. **(b)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

lo que coincide con la ecuación 4.19. Dentro del 7483, las señales P_i son válidas en dos retardos de compuerta, las señales C_i en tres retardos de compuerta, y los Σ_i se producen un retardo de compuerta XOR después de los términos C_i.

Puesto que el 7483 genera cuatro términos de suma, podemos construir un sumador seudoparalelo de n bits a partir de

$$m = \lceil n/4 \rceil \tag{4.28}$$

módulos, donde la notación $\lceil x \rceil$ indica el mínimo entero mayor o igual que x. Así, la unidad resultante tendrá un retardo global en el peor de los casos de tres retardos de compuerta para el acarreo que sale de cada módulo, excepto el último, que tendrá cuatro retardos de compuerta respecto a sus salidas de suma:

$$t_{\text{retardo}} = (3m + 1)t_{\text{compuerta}} \tag{4.29}$$

Los sumadores seudoparalelos son fáciles de construir y tienen una velocidad razonable. Sin embargo, estos diseños no son satisfactorios en ciertas aplicaciones de alta velocidad debido al retardo creciente cuando n aumenta.

4.6.3 Unidades sumadoras de alta velocidad

En el diseño de los circuitos aritméticos, con frecuencia queremos mejorar el desempeño de un circuito aumentando la velocidad con que se pueden realizar las diversas operaciones, como la suma binaria. Podemos mejorar la velocidad de un circuito eligiendo una tecnología en que las compuertas lógicas individuales tengan retardos por propagación más cortos o diseñando el circuito de modo que minimice el número de retardos de compuerta necesarios para realizar la operación. En la mayor parte de los casos, el número de compuertas necesarias para realizar un diseño debe incrementarse para reducir el retardo global por propagación, por lo que hay que tomar una decisión en favor del costo o del desempeño. En las secciones siguientes examinamos varios métodos que sirven para reducir el número de ratardos por propagación de circuitos sumadores en binario. En cada caso evaluamos la cantidad de compuertas necesarias para el diseño y la cantidad total de retardos de compuerta requeridos para obtener la suma de dos números.

Sumadores completamente paralelos

El diseño de sumador más rápido sería estrictamente paralelo. Es decir, todas las entradas se aplicarían en forma simultánea y se propagarían a través de dos niveles de lógica para obtener el resultado. Sin embargo, este método requeriría una gran cantidad de circuitos lógicos y no sería práctico. Consideremos la generación de los tres primeros bits de acarreo de un sumador seudoparalelo:

$$c_0 = x_0 y_0 \tag{4.30}$$

$$\begin{aligned} c_1 &= x_1 y_1 \bar{c}_0 + x_1 y_1 c_0 + x_1 \bar{y}_1 c_0 + \bar{x}_1 y_1 c_0 \\ &= x_1 y_1 + (x_1 \oplus y_1) c_0 \\ &= x_1 y_1 + (x_1 \oplus y_1)(x_0 y_0) \end{aligned} \tag{4.31}$$

$$\begin{aligned} c_2 &= x_2 y_2 + (x_2 \oplus y_2) c_1 \\ &= x_2 y_2 + (x_2 \oplus y_2)[x_1 y_1 + (x_1 \oplus y_1)(x_0 y_0)] \\ &= x_2 y_2 + (x_2 \oplus y_2)(x_1 y_1) + (x_2 \oplus y_2)(x_1 \oplus y_1)(x_0 y_0) \end{aligned} \tag{4.32}$$

Podemos simplificar aún más estas ecuaciones en forma de suma de productos, lo que permite realizar cada una con dos niveles de lógica sea cual sea el ancho

de palabra del sumador. Sin embargo, la cantidad de compuertas aumenta en forma considerable con cada posición significativa de bit.

Sumadores con acarreo en prospectiva

Existen varios términos medios entre las alternativas seudoparalela y estrictamente paralela. Los *sumadores con acarreo en prospectiva* dividen los sumadores completos en grupos y utilizan lógica de derivación de acarreos para acelerar la propagación del acarreo. Es razonable utilizar esta técnica cuando se suman datos numéricos a alta velocidad en intervalos de tiempo fijos. Al examinar las ecuaciones 4.30 a 4.32, definimos los dos términos

$$g_i = x_i y_i \tag{4.33}$$

y

$$p_i = x_i \oplus y_i \tag{4.34}$$

Como estos términos podemos escribir las ecuaciones 4.30 a 4.32 como sigue:

$$c_0 = g_0 \tag{4.35}$$

$$\begin{aligned} c_1 &= g_1 + p_1 c_0 \\ &= g_1 + p_1 g_0 \end{aligned} \tag{4.36}$$

$$\begin{aligned} c_2 &= g_2 + p_2 c_1 \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 \end{aligned} \tag{4.37}$$

Dentro de cada posición de bit, si $g_i = 1$, se *genera* un acarreo, independientemente de la entrada del acarreo c_{i-1}. De manera similar, una entrada de acarreo 1 se *propagará* desde la entrada hasta la salida de la etapa i si $p_i = 1$. Por tanto, los términos g_i y p_i se conocen como términos *generador* y *propagador de acarreo*, respectivamente, para la etapa i. Observe que estos términos se pueden obtener en paralelo para todas las posiciones de bit en un solo retardo de compuerta y que podemos calcular los bits de acarreo en paralelo a partir de los términos generador y propagador en otros dos retardos de compuerta. La figura 4.39a muestra un circuito que realiza las ecuaciones 4.33 y 4.34, mientras que la figura 4.39b muestra un circuito que realiza las ecuaciones 4.35 a 4.37.

El término suma realizado por un sumador completo, definido por la ecuación 4.19, se puede reescribir con base en los términos generador y propagador como sigue:

$$\begin{aligned} s_i &= x_i \oplus y_i \oplus c_{i-1} \\ &= p_i \oplus c_{i-1} \end{aligned} \tag{4.38}$$

Por tanto, una vez disponibles los términos de acarreo, podemos calcular los términos de suma en otro retardo de compuerta más, lo que produce un retardo total del sumador igual a

$$t_{\text{cla}} = 4t_{\text{compuerta}}$$

independiente del ancho de palabra del sumador. Las figuras 4.39a y c muestran los circuitos para un módulo sumador simple y un sumador completo con acarreo en prospectiva de 3 bits. Debe observar que los retardos de las compuertas OR-exclusivo podrían ser casi el doble de los de las compuertas NAND/NOR simples. Por tanto, una estimación más realista del retardo sería

$$4t_{\text{compuerta}} \le t_{\text{cla}} \le 6t_{\text{compuerta}}$$

donde $t_{\text{compuerta}}$ es el retardo a través de una compuerta NAND típica.

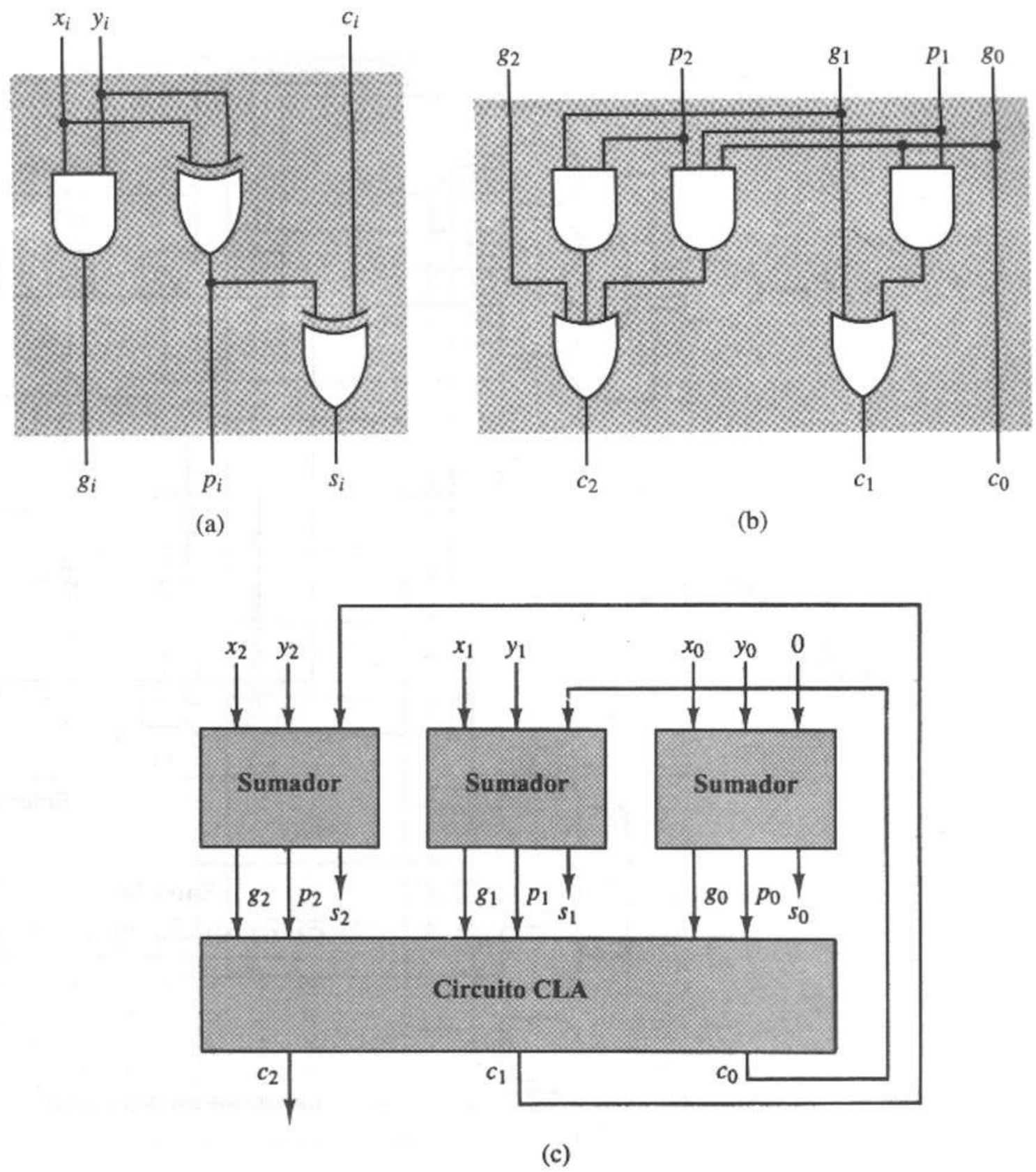


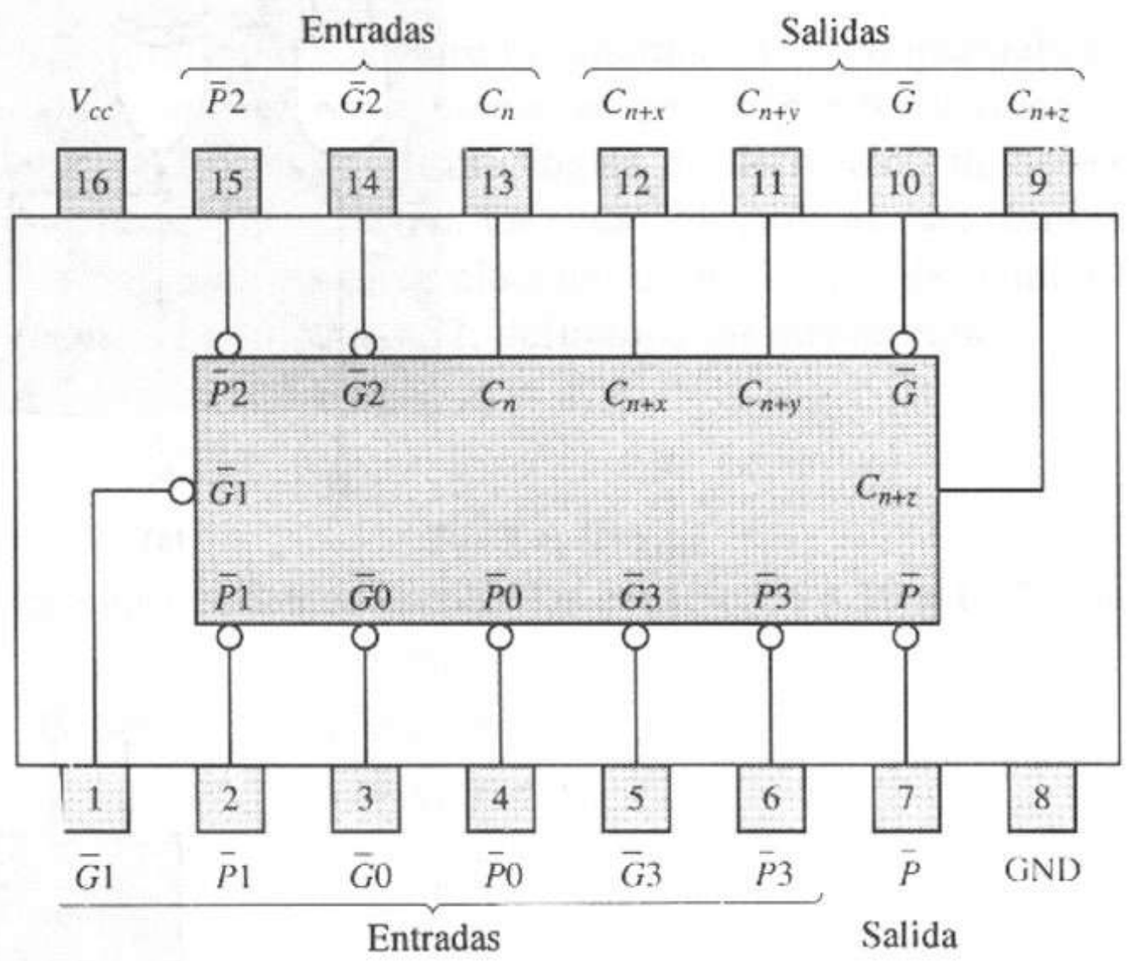
Figura 4.39 Diseño de sumador con acarreo en prospectiva. **(a)** Módulo sumador que produce g_i y p_i. **(b)** Circuito del acarreo en prospectiva (CLA). **(c)** Sumador completo con acarreo en prospectiva.

Generador de acarreo en prospectiva MSI 74182

Los *generadores de acarreo en prospectiva* también están disponibles como módulos MSI estándar, como es el caso del módulo 74182 de la figura 4.40, que produce tres términos de acarreo a partir de cuatro conjuntos de términos P_i y G_i y que puede conectarse en cascada para producir salidas P y G que pueden alimentarse a otros módulos 74182.

Sumadores con detección y conclusión de acarreo

Para las aplicaciones asíncronas, un esquema de suma con *detección y conclusión de acarreo* puede acelerar el proceso de suma de manera notable. Este esquema agrega circuitos lógicos a cada sumador completo, los cuales emiten una señal hacia un circuito de control cuando concluyen la suma. En promedio, los acarreos sólo se propagan un quinto de la longitud de la unidad sumadora; por tanto, en vez de esperar el retardo por propagación de peor caso



(a)

Entradas $\bar{G}3$	$\bar{G}2$	$\bar{G}1$	$\bar{G}0$	$\bar{P}3$	$\bar{P}2$	$\bar{P}1$	Salida $\bar{G}$
L	×	×	×	×	×	×	L
×	L	×	×	L	×	×	L
×	×	L	×	L	L	×	L
×	×	×	L	L	L	L	L
Las demás combinaciones							H

(b)

Entradas $\bar{P}3$	$\bar{P}2$	$\bar{P}1$	$\bar{P}0$	Salida $\bar{P}$
L	L	L	L	L
Las demás combinaciones				H

(c)

Entradas $\bar{G}0$	$\bar{P}0$	C_n	Salida C_{n+x}
L	×	×	H
×	L	H	H
Las demás combinaciones			L

(d)

Entradas $\bar{G}1$	$\bar{G}0$	$\bar{P}1$	$\bar{P}0$	C_n	Salida C_{n+y}
L	×	×	×	×	H
×	L	L	×	×	H
×	×	L	L	H	H
Las demás combinaciones					L

(e)

Entradas $\bar{G}2$	$\bar{G}1$	$\bar{G}0$	$\bar{P}2$	$\bar{P}1$	$\bar{P}0$	C_n	Salida C_{n+z}
L	×	×	×	×	×	×	H
×	L	×	L	×	×	×	H
×	×	L	L	L	×	×	H
×	×	×	L	L	L	H	H
Las demás combinaciones							L

(f)

Figura 4.40 El generador 74182 de acarreo en prospectiva. **(a)** Configuración de *pines* del paquete. **(b)** Tabla funcional para la salida $\bar{G}$. **(c)** Tabla funcional para la salida $\bar{P}$. **(d)** Tabla funcional para la salida C_{n+x}. **(e)** Tabla funcional para la salida C_{n+y}. **(f)** Tabla funcional para la salida C_{n+z}. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

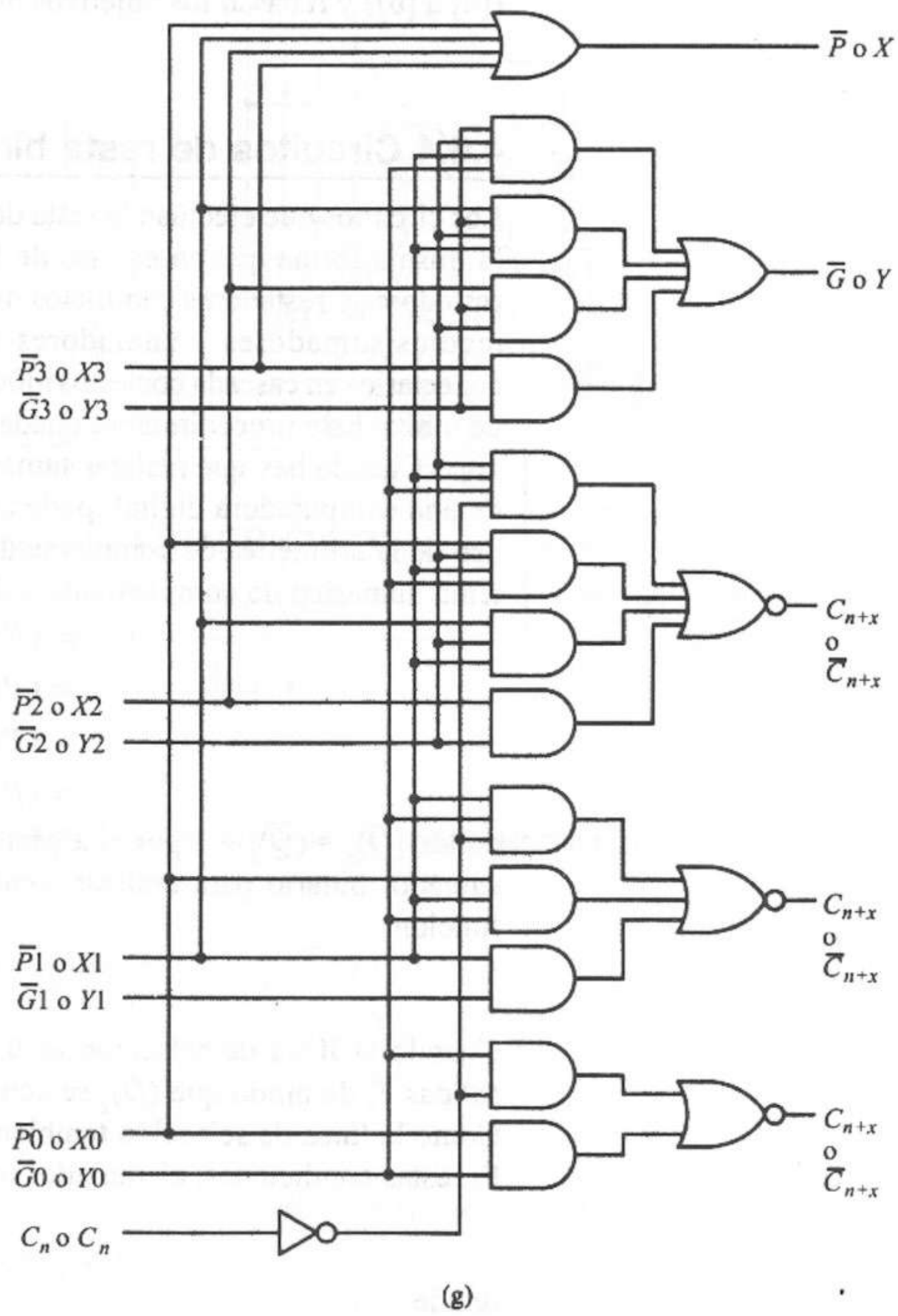


Figura 4.40 (continuación) El generador 74182 de acarreo en prospectiva. **(g)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

cada vez que se sumen dos números, la señal de conclusión del acarreo permite que comiencen nuevas sumas tan pronto como se haya realizado la última.

Sumadores que guardan el acarreo

Otra técnica de aceleración es de utilidad cuando la cadena de números debe totalizarse, o acumularse. Una técnica para guardar el acarreo inhibe la propagación del acarreo, guardando los acarreos entre las etapas en elementos de almacenamiento llamados *flip-flops*. Después, en la última suma, se permite que los acarreos se propaguen de manera seudoparalela. Presentaremos los circuitos con *flip-flops* en el capítulo 6.

Los detalles completos acerca de estas técnicas están en la bibliografía ([4] a [6]) y rebasan los objetivos de este capítulo.

4.6.4 Circuitos de resta binaria

Los circuitos que efectúan la resta de números binarios se pueden desarrollar de la misma forma que en el caso de la suma binaria. Podemos diseñar medios-restadores y restadores completos mediante el procedimiento ilustrado para los medios-sumadores y sumadores completos de la figura 4.35, y después conectarlos en cascada con estos módulos para obtener restadores seudoparalelos de n bits. Este procedimiento queda como ejercicio para el lector.

Cuando hay que realizar sumas y restas, como en los circuitos aritméticos de una computadora digital, podemos simplificar el diseño global mediante el uso de la aritmética de complemento a dos. Recordemos que la resta en el sistema numérico de complemento a dos se realiza de la manera siguiente:

$$\begin{aligned}(R)_2 &= (P)_2 - (Q)_2\\ &= (P)_2 + (-Q)_2\\ &= (P)_2 + [Q]_2\\ &= (P)_2 + (\bar{Q})_2 + 1\end{aligned}$$

donde $[\overline{Q}]_2 = (\overline{Q})_2 + 1$ por el algoritmo 1.4. La figura 4.41 ilustra el uso de un sumador binario para realizar sumas y restas. El módulo sumador realiza la función

$$(\Sigma)_2 = (A)_2 + (B)_2 + C_0 \tag{4.39}$$

Cuando la línea de selección es 0, el multiplexor guía sus entradas A a sus salidas Y, de modo que $(Q)_2$ se conecta a las entradas B del módulo sumador. Como la línea de selección también está conectada a $C0$ del sumador, $C0 = 0$. En estas condiciones, el sumador realiza la siguiente operación:

$$(\Sigma)_2 = (A)_2 + (B)_2 + C0$$

donde

$$\begin{aligned}(A)_2 &= (P)_2\\ (B)_2 &= (Q)_2\\ C0 &= 0\end{aligned}$$

Así,

$$\begin{aligned}(R)_2 &= (\Sigma)_2\\ &= (P)_2 + (Q)_2 + 0\\ &= (P)_2 + (Q)_2\end{aligned}$$

Ésta es la operación de suma.

Ahora, consideremos el caso en que $S = 1$. Con $S = 1$, el módulo multiplexor guía sus entradas B a sus salidas Y, de modo que el complemento de $(Q)_2$ se conecta a las entradas B del módulo sumador. La línea de selección S también controla la salida $C0$ del módulo sumador, de modo que

$$\begin{aligned}(R)_2 &= (\Sigma)_2\\ &= (A)_2 + (B)_2 + C0\end{aligned}$$

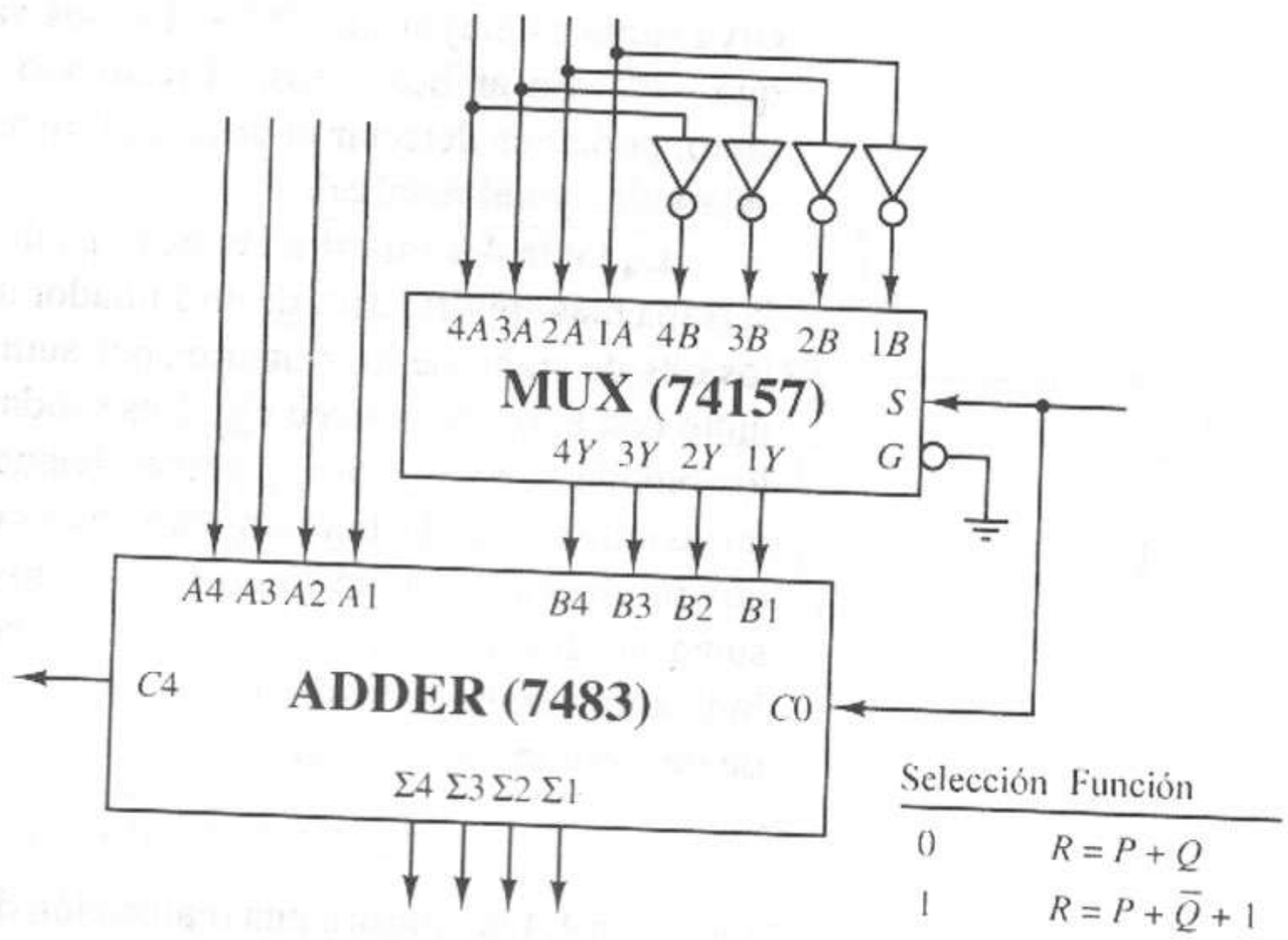


Figura 4.41 Sumador/restador de complemento a dos.

donde

$$(A)_2 = (P)_2$$
$$(B)_2 = (\bar{Q})_2$$
$$C0 = 1$$

Así,

$$\begin{aligned} R &= (P)_2 + (\bar{Q})_2 + 1 \\ &= (P)_2 + [Q]_2 \\ &= (P)_2 - (Q)_2 \end{aligned}$$

Ésta es la operación de resta.

Por tanto, podemos implantar las funciones de suma y resta con un único módulo sumador y un multiplexor, como lo muestra la figura 4.41.

4.6.5 Detección de un desbordamiento aritmético

Como vimos en el capítulo 1, el intervalo de valores que podemos representar mediante números de n bits en el sistema numérico de complemento a dos es

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$

Cualquier operación aritmética que intente generar un valor fuera de este intervalo producirá una condición de *desbordamiento*. En estos casos, el número resultante de n bits no es una representación válida del resultado; por tanto, hay que detectar todas las condiciones de desbordamiento de modo que no se utilicen en forma inadvertida resultados no válidos.

En el capítulo 1 mostramos que, en la aritmética de complemento a dos, se tienen condiciones de desbordamiento cuando se suman dos valores positivos cuya suma es mayor que $2^{n-1} - 1$ o dos valores negativos cuya suma es menor que -2^{n-1}. En ambos casos, el resultado tiene un bit de signo incorrecto. Por tanto, podemos detectar el desbordamiento observando los bits de signo de los operandos y del resultado.

La tabla 4.3 muestra las ocho posibles situaciones que pueden ocurrir en la etapa más significativa de un sumador de n bits. Los bits a_{n-1} y b_{n-1} representan los bits de signo de los números por sumar, así que son entradas de esta etapa, junto con el bit de acarreo c_{n-2}. Las salidas de la etapa son el acarreo de salida y los bits de suma c_{n-1} y s_{n-1}, respectivamente. Como vemos en la tabla, ocurre una condición de desbordamiento en dos casos: la suma de dos valores positivos que producen un bit de signo $s_{n-1} = 1$, lo que indica un resultado negativo, y la suma de dos valores negativos que produce un bit de signo $s_{n-1} = 0$, lo que indica un resultado positivo. Por tanto, una expresión lógica para la condición de desbordamiento V es

$$V = \bar{a}_{n-1}\bar{b}_{n-1}s_{n-1} + a_{n-1}b_{n-1}\bar{s}_{n-1} \tag{4.40}$$

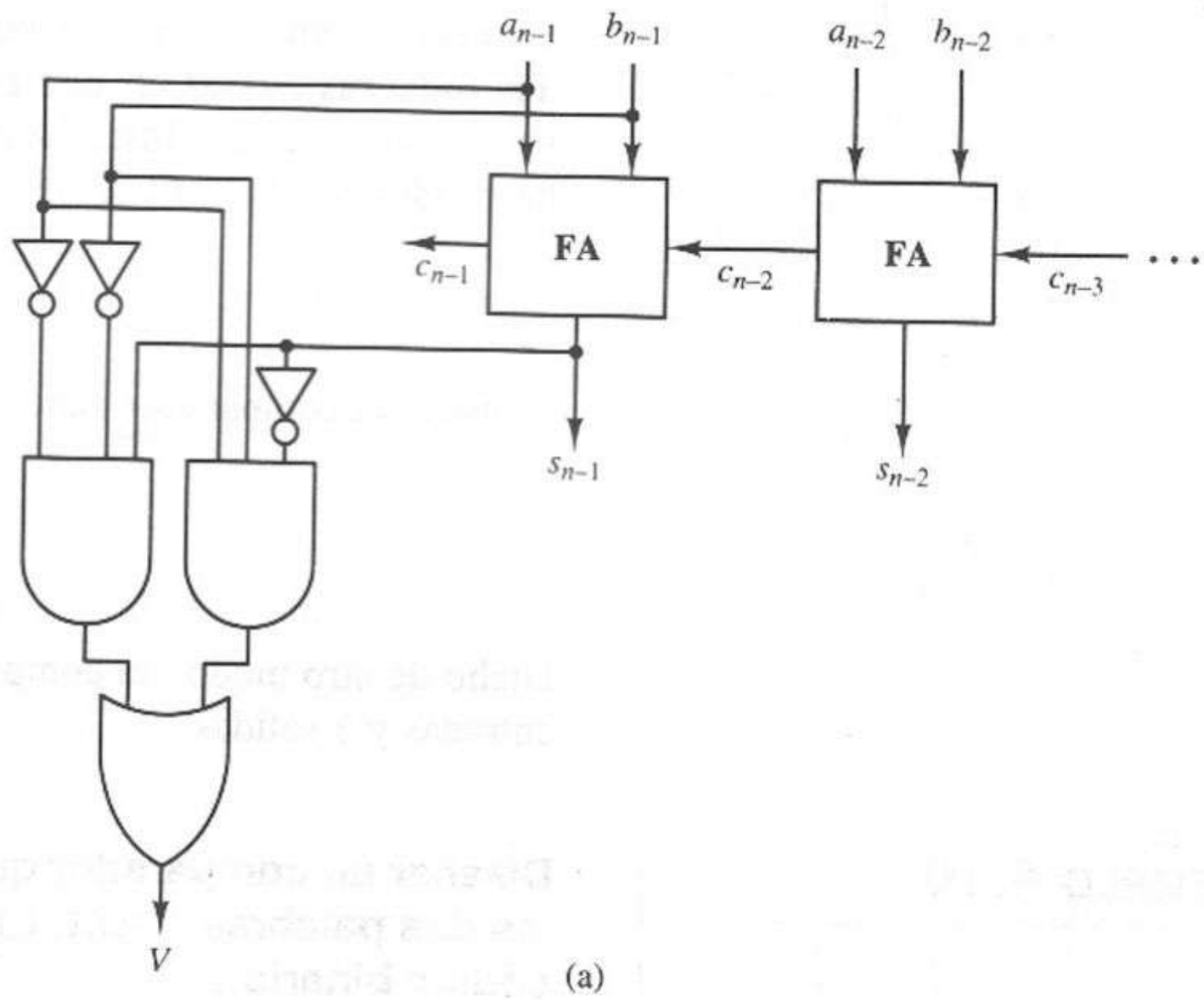
La figura 4.42a muestra una realización de la ecuación 4.40 mediante un circuito lógico AND/OR.

TABLA 4.3 ETAPA MÁS SIGNIFICATIVA DE UN SUMADOR DE n BITS

Entradas del sumador			Salidas del sumador		Desbordamiento
a_{n-1}	b_{n-1}	c_{n-2}	c_{n-1}	s_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

También podemos detectar los desbordamientos observando los bits de acarreo de entrada y de acarreo de salida, c_{n-2} y c_{n-1}, respectivamente, de la etapa más significativa del sumador completo. Si analizamos las dos filas de la tabla 4.3 donde ocurren los desbordamientos, podemos ver que éstas son las únicas dos situaciones en que $c_{n-2} \neq c_{n-1}$. Como vemos en las dos primeras filas de la tabla 4.3, la suma de dos números positivos siempre produce una salida de acarreo $c_{n-1} = 0$, así que la entrada de acarreo al bit más significativo, c_{n-2}, también debe ser 0 para producir una suma positiva correcta. Un acarreo de 1 en este bit produce $s_{n-1} = 1$, lo cual es incorrecto. En la fila 7 de la tabla 4.3 aparece una situación similar. La suma de dos números negativos produce $c_{n-1} = 1$. La ausencia de acarreo en este bit produce un bit de signo incorrecto $s_{n-1} = 0$. Así, la condición de desbordamiento queda señalada por $c_{n-2} \neq c_{n-1}$ o $c_{n-2} \oplus c_{n-1} = 1$. Por tanto,

$$V = c_{n-2} \oplus c_{n-1} \tag{4.41}$$



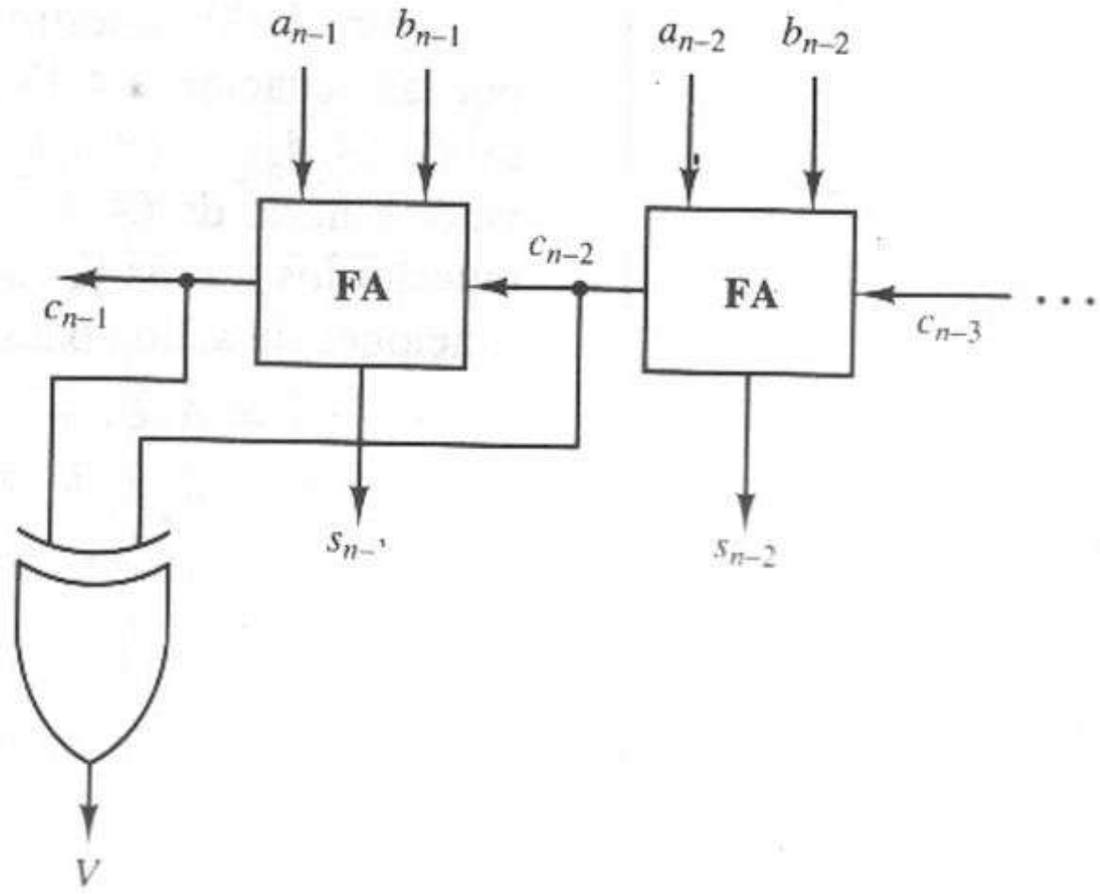


Figura 4.42 Detección de desbordamiento con complemento a dos. **(a)** Uso de bits de signo. **(b)** Uso de bits de acarreo.

El circuito lógico correspondiente requiere una única compuerta XOR, como se aprecia en la figura 4.42b. Este circuito es más sencillo que el de la figura 4.42a, pero requiere un acceso al bit de acarreo c_{n-2} entre las dos últimas etapas del sumador. Este bit no siempre está accesible, como ocurre cuando utilizamos los módulos sumadores MSI 7482 o 7483 y varios otros diseños de sumador paralelo. En estos casos, los desbordamientos se detectan mediante el análisis del bit de signo, usando la ecuación 4.40.

4.7 Comparadores

Un *comparador* es un dispositivo aritmético que determina la magnitud relativa de dos números binarios y que tiene aplicaciones en diversos sistemas digitales. En general, un comparador puede realizar una comparación de magnitud de dos palabras A y B en códigos binario o BCD. Se toman tres decisiones completamente decodificadas acerca de las dos palabras, y están disponibles en las salidas; a saber, $A > B$, $A < B$ y $A = B$. Ver la figura 4.43a. Si

$$A = (A_{n-1}A_{n-2}\cdots A_0)_2 \qquad (4.42)$$
$$B = (B_{n-1}B_{n-2}\cdots B_0)_2$$

entonces el comparador generará tres señales de salida, como sigue:

$$f_1 = 1, \quad \text{si } A < B$$
$$f_2 = 1, \quad \text{si } A = B \qquad (4.43)$$
$$f_3 = 1, \quad \text{si } A > B$$

Dicho de otro modo, un comparador es un módulo lógico combinatorio de $2n$ entradas y 3 salidas.

EJEMPLO 4.10

Diseñar un comparador que compare las dos palabras $A = (A_1A_0)_2$ y $B = (B_1B_0)_2$ en código binario.

La figura 4.43b muestra la tabla de verdad para las señales de salida definidas por las ecuaciones 4.43 y la figura 4.43c muestra el mapa K de la función de salida $(A_1A_0)_2 < (B_1B_0)_2$. Observe que aparece un 1 en cada bloque donde el valor binario de $(A_1A_0)_2$ es menor que el de $(B_1B_0)_2$. La figura 4.43c también muestra los mapas K para las salidas $(A_1A_0)_2 = (B_1B_0)_2$ y $(A_1A_0)_2 > (B_1B_0)_2$. Las funciones de salida para estos mapas son

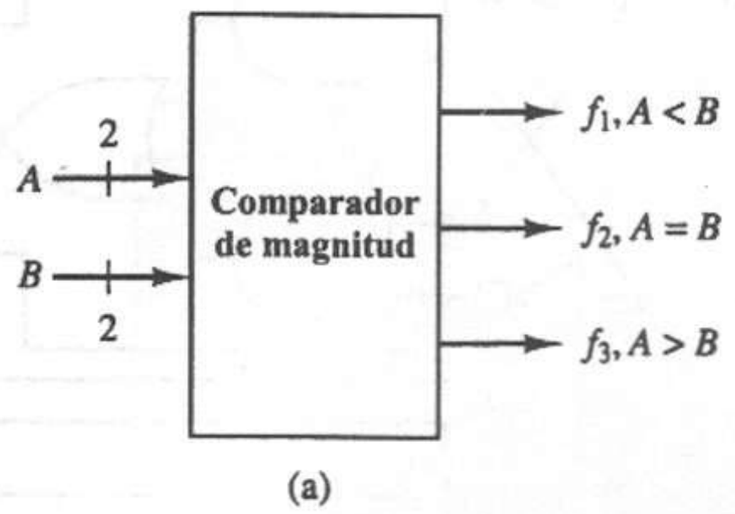
$$f_1 = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0 \quad \text{para } (A_1A_0)_2 < (B_1B_0)_2$$
$$f_2 = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0$$
$$+A_1\bar{A}_0B_1\bar{B}_0 + A_1A_0B_1B_0 \quad \text{para } (A_1A_0)_2 = (B_1B_0)_2$$
$$f_3 = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0 \quad \text{para } (A_1A_0)_2 > (B_1B_0)_2$$

La figura 4.44 muestra una realización de este circuito con compuertas lógicas.

7485

Un comparador de magnitudes de 4 bits similar al obtenido en el ejemplo 4.10 es el circuito 7485 que aparece en la figura 4.45. Observe que este circuito se puede poner en forma de diagrama de bloques como en la figura 4.46a. Las entradas de datos de este módulo son

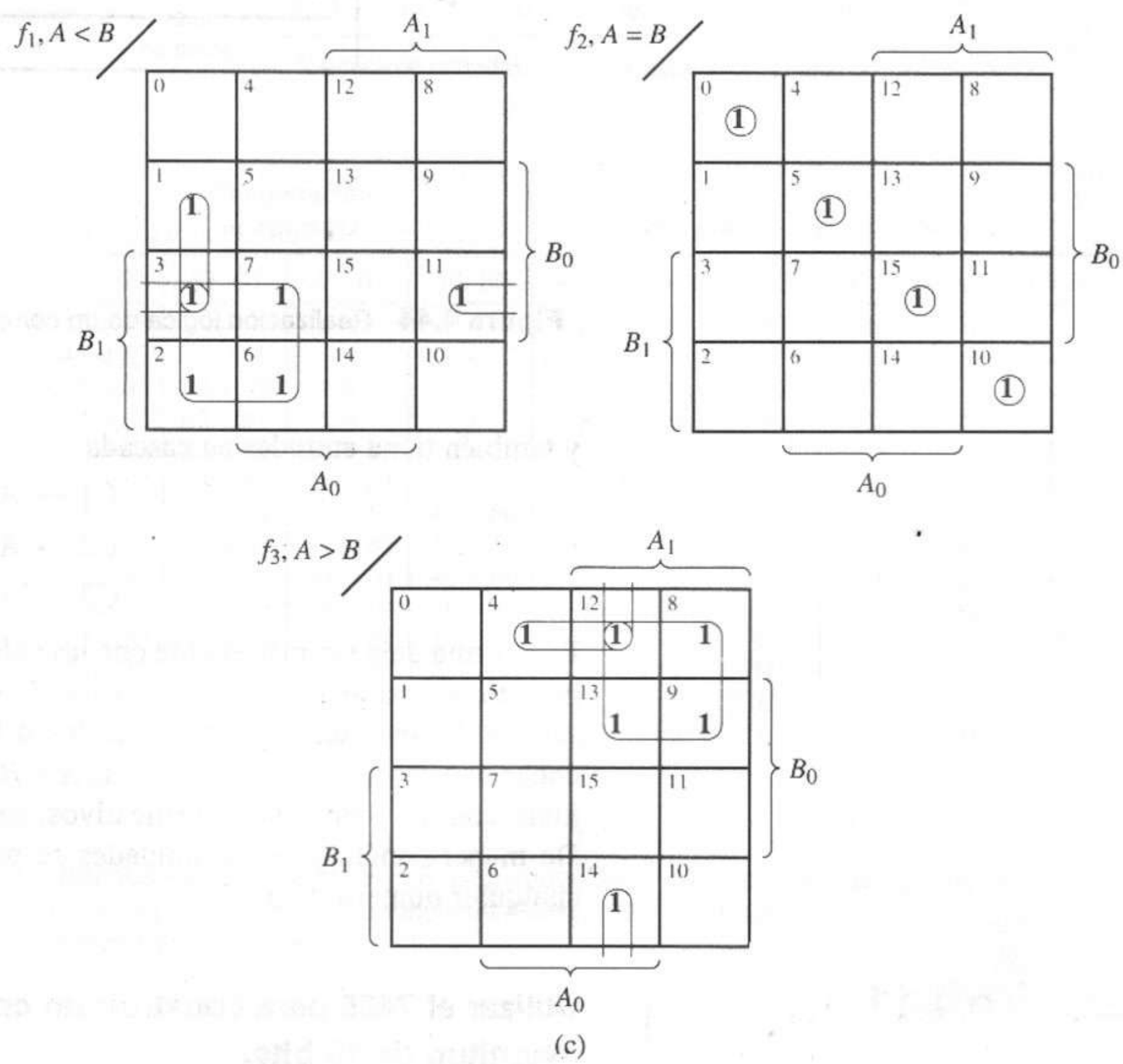
$$A = (A_3, A_2, A_1, A_0)_2 \qquad (4.44)$$
$$B = (B_3, B_2, B_1, B_0)_2$$



(a)

i	A_1	A_0	B_1	B_0	f_1	f_2	f_3
0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0
2	0	0	1	0	1	0	0
3	0	0	1	1	1	0	0
4	0	1	0	0	0	0	1
5	0	1	0	1	0	1	0
6	0	1	1	0	1	0	0
7	0	1	1	1	1	0	0
8	1	0	0	0	0	0	1
9	1	0	0	1	0	0	1
10	1	0	1	0	0	1	0
11	1	0	1	1	1	0	0
12	1	1	0	0	0	0	1
13	1	1	0	1	0	0	1
14	1	1	1	0	0	0	1
15	1	1	1	1	0	1	0

(b)



(c)

Figura 4.43 Diseño de comparador de dos bits. **(a)** Diagrama funcional. **(b)** Tabla de verdad. **(c)** Mapas K.

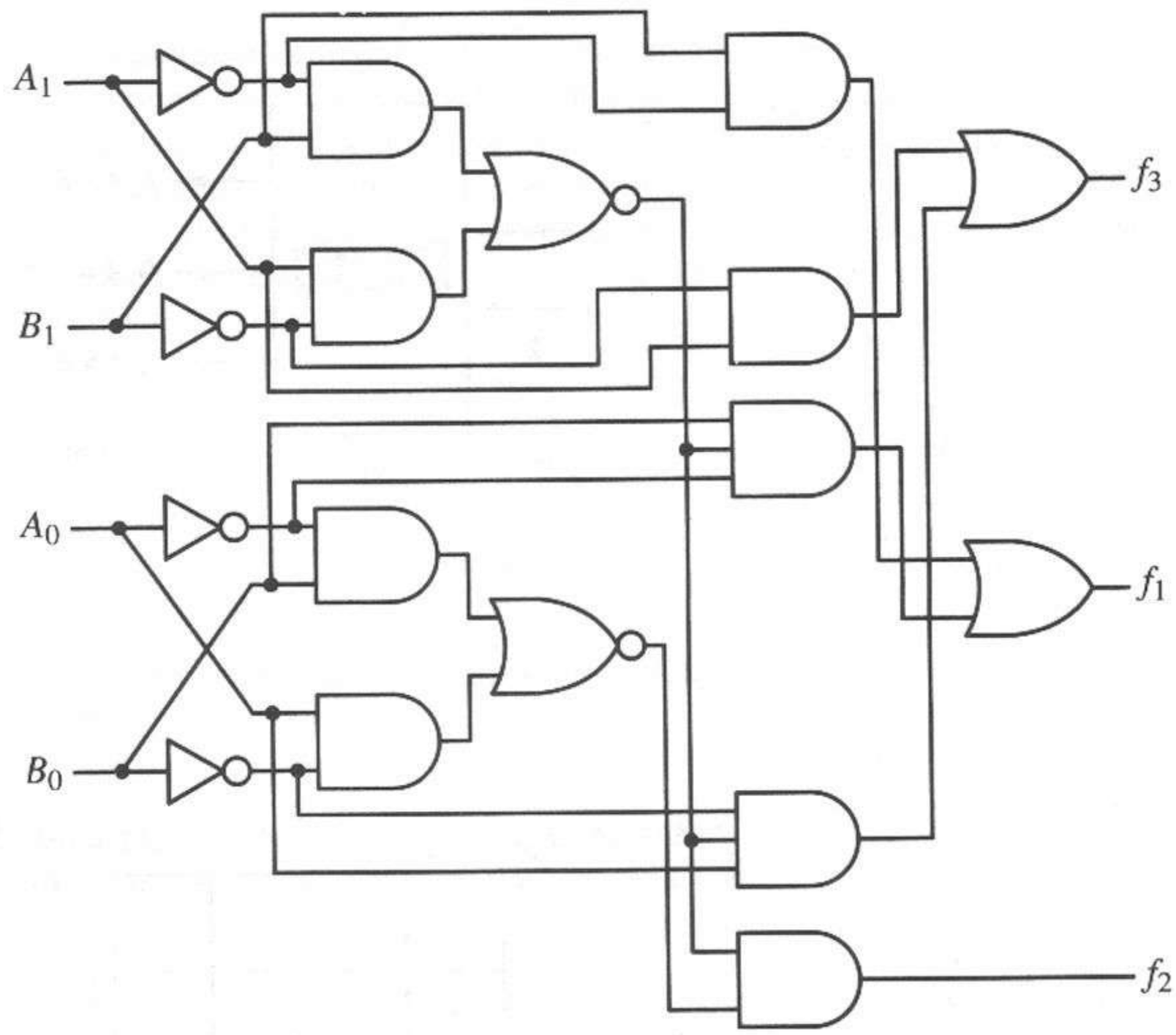


Figura 4.44 Realización lógica de un comparador de dos bits.

y también tiene entradas en cascada

$$\begin{aligned} C1 &\rightarrow A < B \\ C2 &\rightarrow A = B \\ C3 &\rightarrow A > B \end{aligned} \tag{4.45}$$

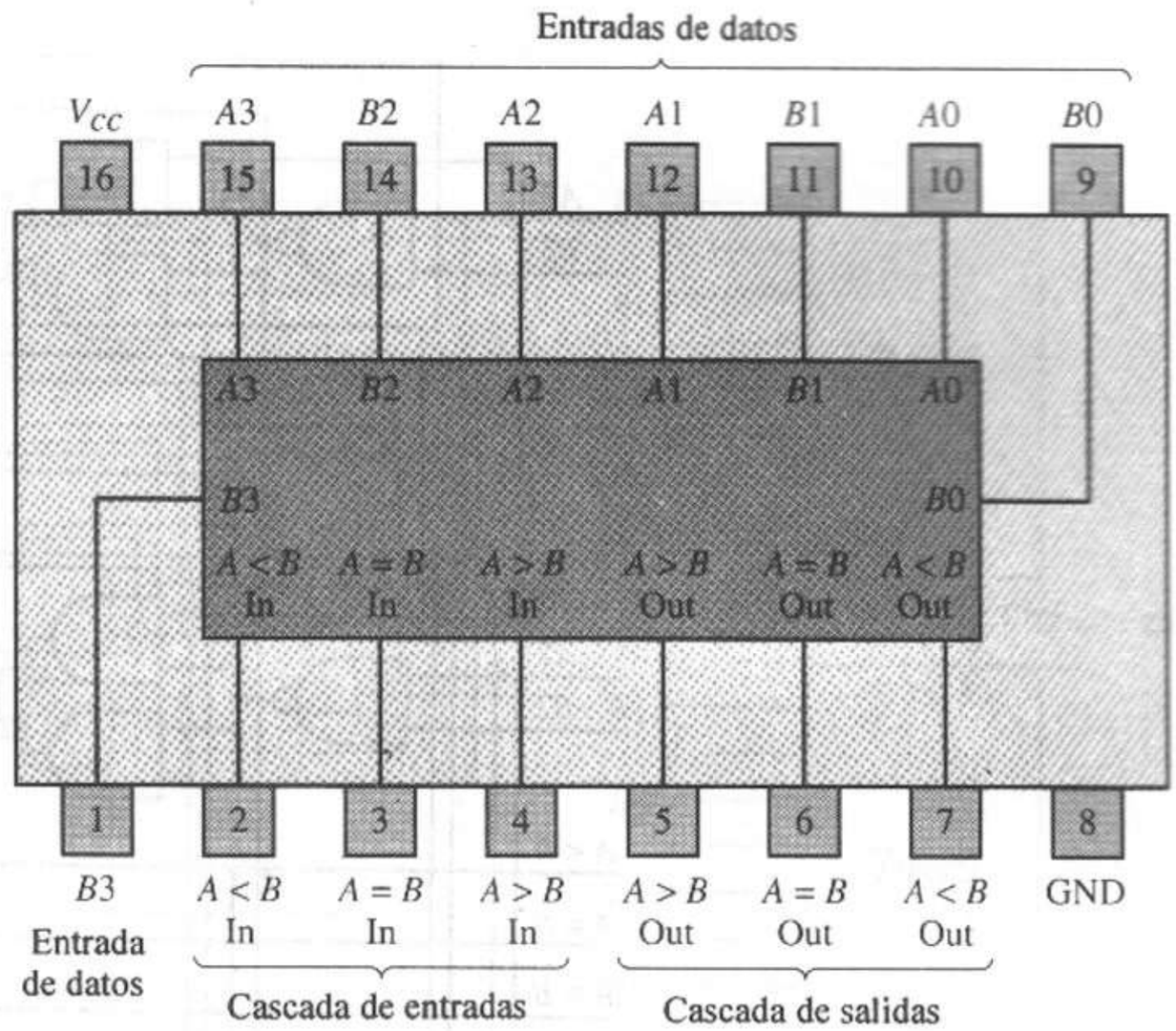
Esta forma deja ver claramente que las palabras de longitud mayor que 4 bits se pueden comparar formando una cascada con estas unidades. Por ejemplo, las salidas de una etapa que controla los 4 bits menos significativos se pueden conectar a las terminales de entrada $A < B$, $A = B$ y $A > B$ de la siguiente etapa, junto con los 4 bits más significativos, para generar un comparador de 8 bits. De manera análoga, estas unidades se pueden ampliar completamente hasta cualquier número de bits.

EJEMPLO 4.11

Utilizar el 7485 para construir un comparador de magnitud de 16 bits.

Podemos conectar en cascada cuatro módulos 7485, como se muestra en la figura 4.46b para obtener un comparador de 16 bits. Observe que las condiciones iniciales en el primer estado deben ser

$$(C_1, C_2, C_3) = (0, 1, 0) \tag{4.46}$$



(a)

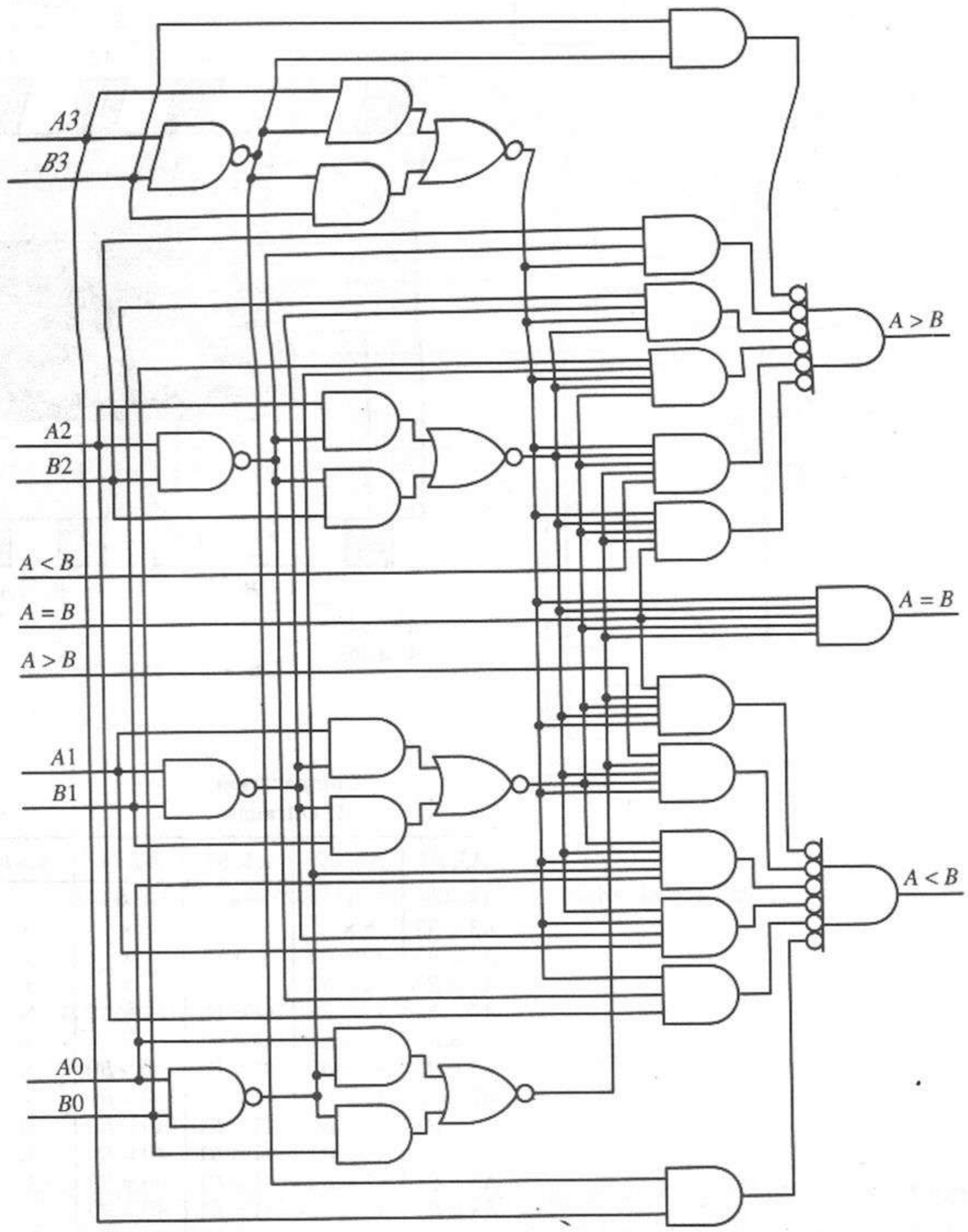
Comparación de entradas				Cascada de entradas			Salidas		
$A3, B3$	$A2, B2$	$A1, B1$	$A0, B0$	$A > B$	$A < B$	$A = B$	$A > B$	$A < B$	$A = B$
$A3 > B3$	×	×	×	×	×	×	H	L	L
$A3 < B3$	×	×	×	×	×	×	L	H	L
$A3 = B3$	$A2 > B2$	×	×	×	×	×	H	L	L
$A3 = B3$	$A2 < B2$	×	×	×	×	×	L	H	L
$A3 = B3$	$A2 = B2$	$A1 > B1$	×	×	×	×	H	L	L
$A3 = B3$	$A2 = B2$	$A1 < B1$	×	×	×	×	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 > B0$	×	×	×	H	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 < B0$	×	×	×	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	H	L	L	H	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	H	L	L	H	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	L	H	L	L	H
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	×	×	H	L	L	H
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	H	H	L	L	L	L
$A3 = B3$	$A2 = B2$	$A1 = B1$	$A0 = B0$	L	L	L	H	H	L

(b)

Figura 4.45 El comparador de magnitudes 7485 de cuatro bits. **(a)** Configuración de *pines* del paquete. **(b)** Tabla funcional. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

lo que indica que $A = B$. La salida del circuito es f_1, f_2, f_3 a partir de la etapa más significativa. El circuito compara los números

$$A = (A_{15}, A_{14}, \ldots, A_0)_2 \qquad (4.47)$$
$$B = (B_{15}, B_{14}, \ldots, B_0)_2$$



(c)

Figura 4.45 (continuación) El comparador de magnitudes 7485 de cuatro bits. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

4.8 Ejemplo de diseño: Una unidad aritmética-lógica de computadora

Los diseños de la mayor parte de los sistemas de cómputo combinan las operaciones aritméticas y lógicas que describimos en el capítulo 2 y en una sección anterior de este capítulo en una única unidad funcional llamada *unidad aritmética-lógica*, o ALU. El símbolo lógico estándar de una ALU aparece en la

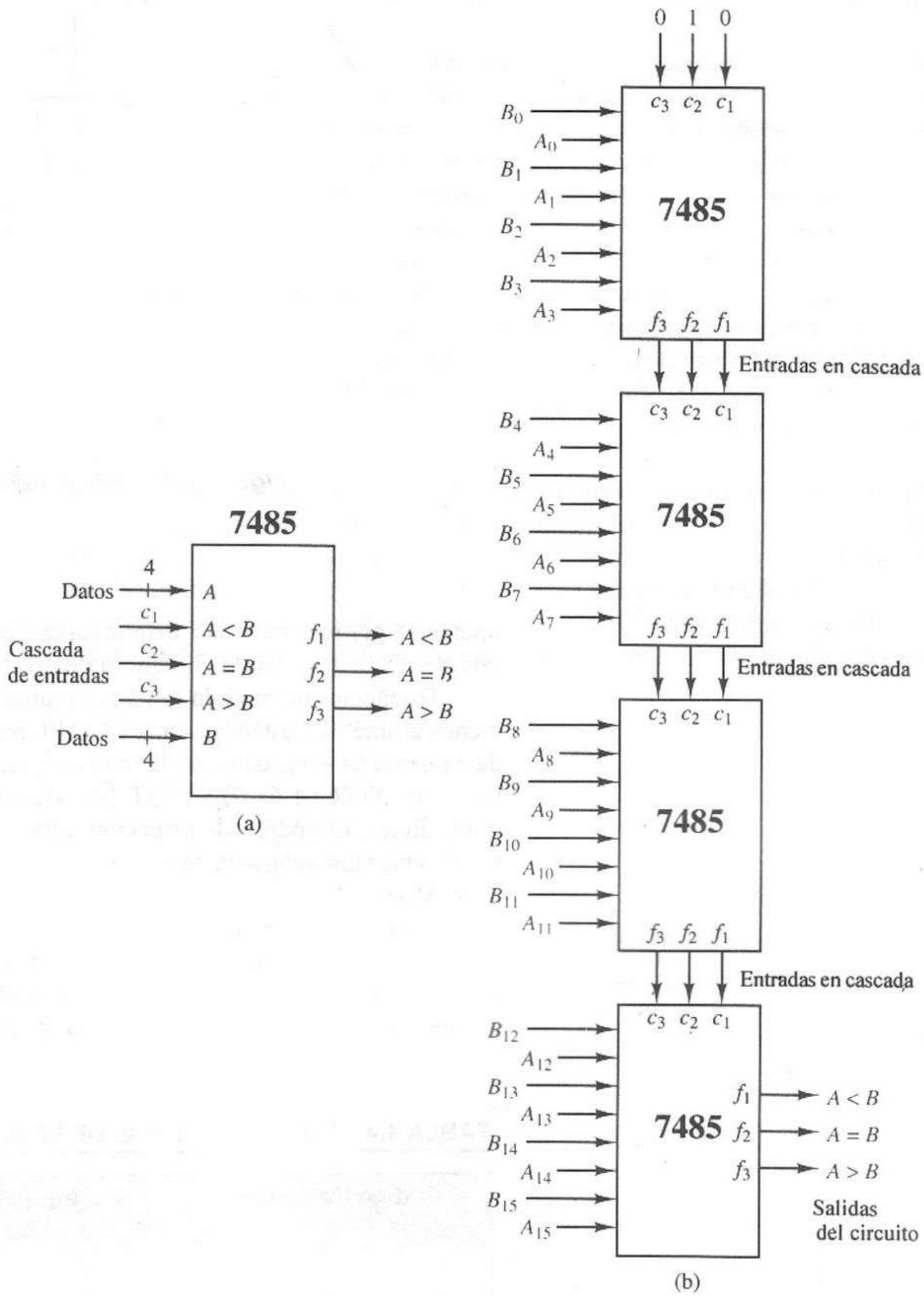


Figura 4.46 Cascada con el comparador de magnitudes 7485 de cuatro bits. **(a)** Diagrama funcional. **(b)** Comparador de 16 bits.

figura 4.47. La salida de la ALU es un número binario de n bits, $F = (f_{n-1} \ldots f_0)_2$, que es el resultado obtenido al realizar cierta operación aritmética o lógica sobre dos números binarios de n bits, u operandos, $A = (a_{n-1} \ldots a_0)_2$ y $B = (b_{n-1} \ldots b_0)_2$. La

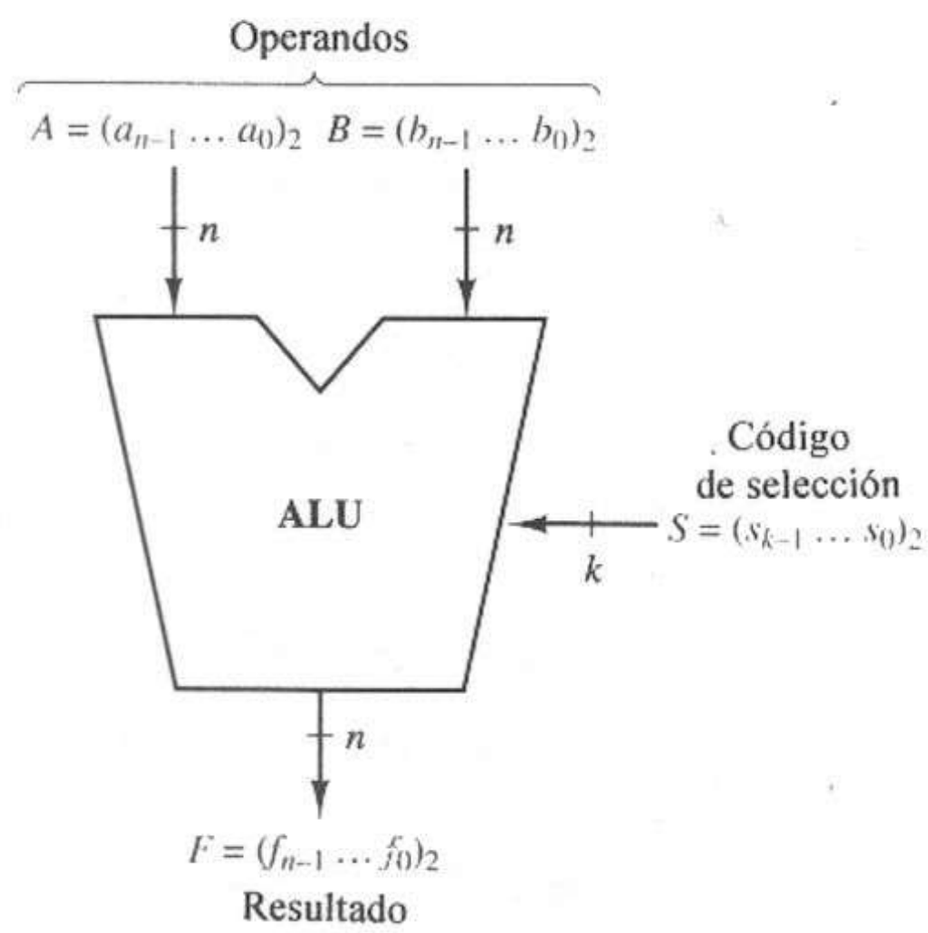


Figura 4.47 Símbolo lógico de una ALU.

operación por realizar queda determinada mediante un código de selección de k bits $S = (S_{k-1} \ldots S_0)$, donde el número de posibles operaciones de la ALU es 2^k.

Diseñemos un módulo ALU que realice ocho funciones: las cuatro operaciones aritméticas estándar suma $(A + B)$, resta $(A - B)$, incremento $(A + 1)$ y decremento $(A - 1)$, así como las cuatro operaciones lógicas AND $(A \cap B)$, OR $(A \cup B)$, XOR $(A \oplus B)$ y NOT $(\overline{A})$. Puesto que tenemos un total de ocho operaciones, el código de selección debe tener 3 bits; es decir, $S = S_2S_1S_0$. Definiremos los códigos de selección como en la tabla 4.4 para las ocho funciones de la ALU.

Queremos desarrollar el diseño de manera *jerárquica, descendente*. Esto significa que debemos descomponer el diseño de nivel superior de la ALU inicialmente en varios módulos. Estos módulos se descomponen a su vez, hasta que todo el diseño se puede representar mediante una jerarquía interconectada

TABLA 4.4 TABLA FUNCIONAL DE LA ALU

Código de selección			Función ALU	Descripción
S_2	S_1	S_0		
0	0	0	$F = A + B$	Suma
0	0	1	$F = A - B$	Resta
0	1	0	$F = A + 1$	Incremento
0	1	1	$F = A - 1$	Decremento
1	0	0	$F = A \cap B$	AND
1	0	1	$F = A \cup B$	OR
1	1	0	$F = \bar{A}$	NOT
1	1	1	$F = A \oplus B$	XOR

de módulos funcionales pequeños y bien definidos. A continuación, diseñamos, implantamos y verificamos en forma individual los circuitos lógicos para estos módulos. Por último, los módulos se conectan entre sí para formar el circuito ALU completo.

El intervalo deseado de números que debe manejar la ALU para una aplicación dada determina el número de bits, n, de los números binarios A, B y F. Para facilitar el desarrollo del diseño de un circuito ALU que se pueda utilizar para valores arbitrarios de n, comenzaremos nuestro diseño descendente descomponiendo nuestra ALU en secciones de 1 bit, donde la sección i realiza las funciones deseadas sobre los bits a_i y b_i de los operandos y produce el bit de resultado f_i, como se muestra en la figura 4.48a. Para las funciones aritméticas, observe que cada sección tiene una entrada de acarreo C_{i-1} y una salida de acarreo C_i. Una vez que hayamos diseñado el circuito para la sección básica de un bit, podremos crear una ALU de n bits (es decir, una ALU para la que A, B y F sean números de n bits) conectando en cascada n secciones de 1 bit, como en la figura 4.48b, con un circuito especial para generar el acarreo inicial en la entrada C_{-1}.

Ahora, consideremos el diseño de una sección básica de 1 bit. Como las cuatro operaciones aritméticas están relacionadas, al igual que las cuatro operaciones lógicas, podemos dividir la sección de ALU en tres módulos: una unidad aritmética (AU), una unidad lógica (LU) y un multiplexor de salida. La figura 4.49 ilustra esto con un diagrama de bloques. Los códigos de selección de la tabla 4.4 se definen de modo que el bit S_2 determina si la salida f_i debe ser

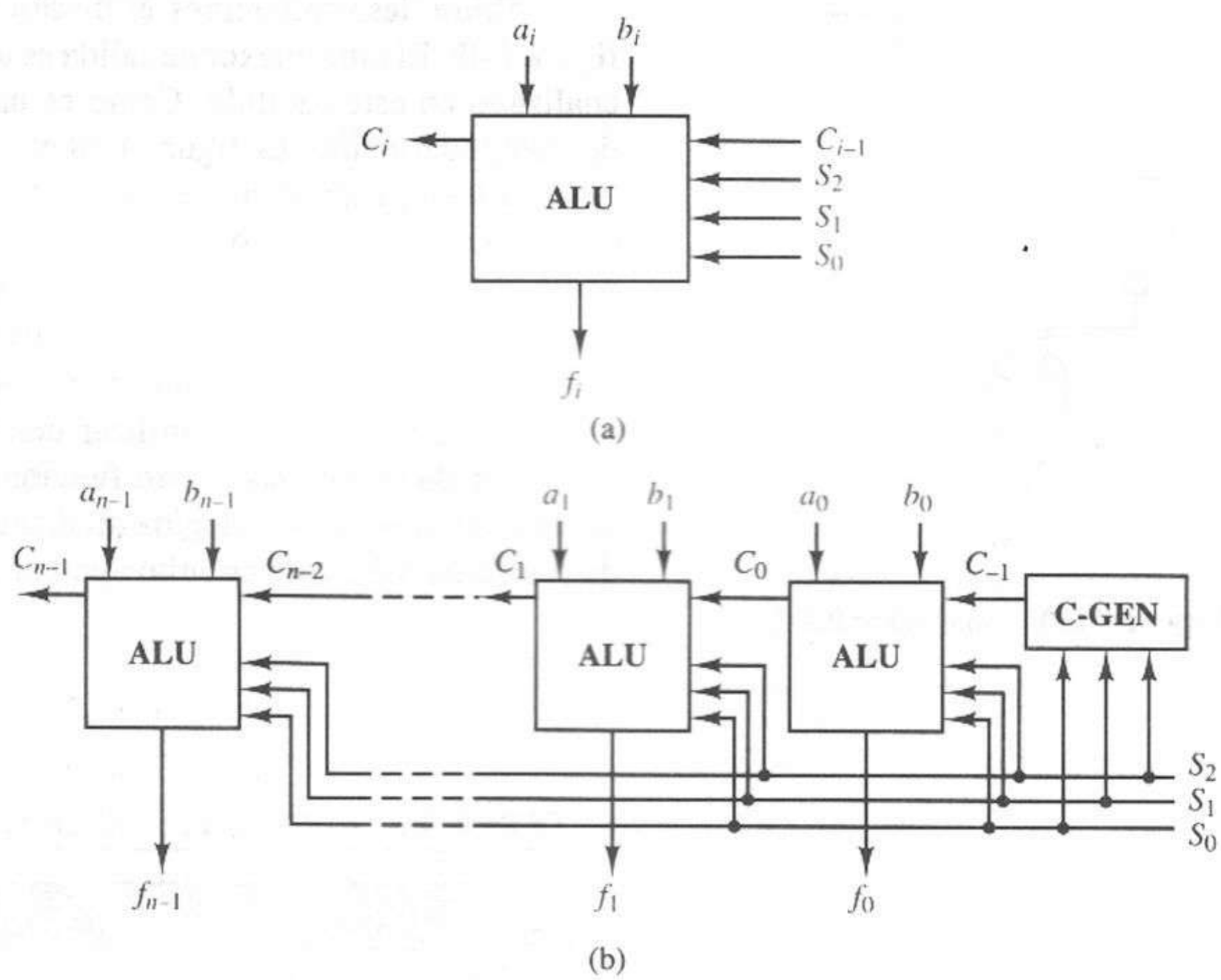


Figura 4.48 Partición de una ALU en secciones de 1 bit. **(a)** Sección de ALU de 1 bit. **(b)** ALU de n bits como cascada de n secciones de 1 bit.

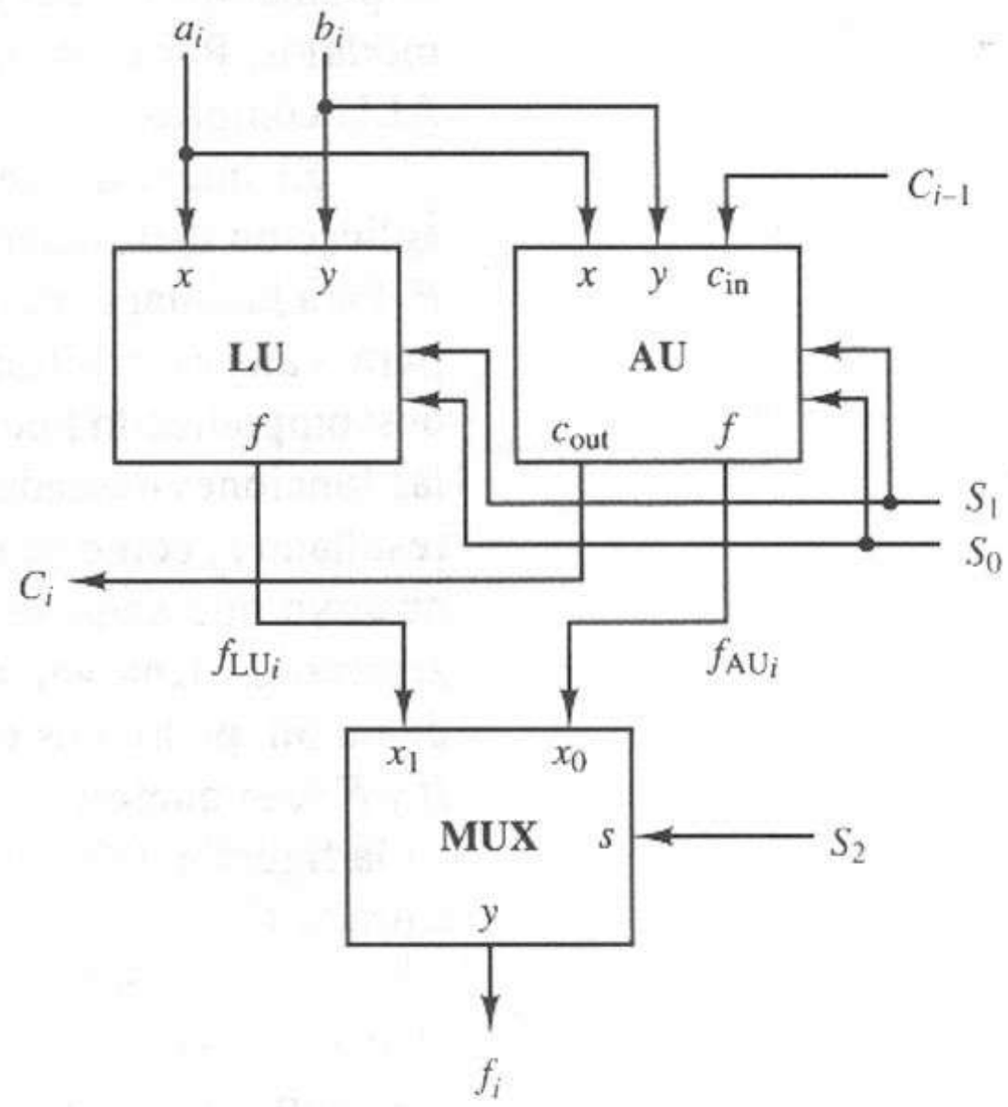


Figura 4.49 ALU de un bit dividida en unidades aritmética y lógica separadas.

un resultado aritmética o lógico. Por tanto, el multiplexor de salida selecciona la salida AU ($f_i = f_{AU_i}$) para $S_2 = 0$, y la salida LU ($f_i = f_{LU_i}$) para $S_2 = 1$.

Ahora desarrollaremos el diseño de cada uno de los tres módulos de la figura 4.49. El multiplexor de salida es un módulo multiplexor 2 a 1 estándar, ya analizado en este capítulo. Como es un diseño sencillo y directo, no hay que descomponerlo más. La figura 4.50 muestra un circuito con compuertas NAND de dos niveles para el multiplexor 2 a 1.

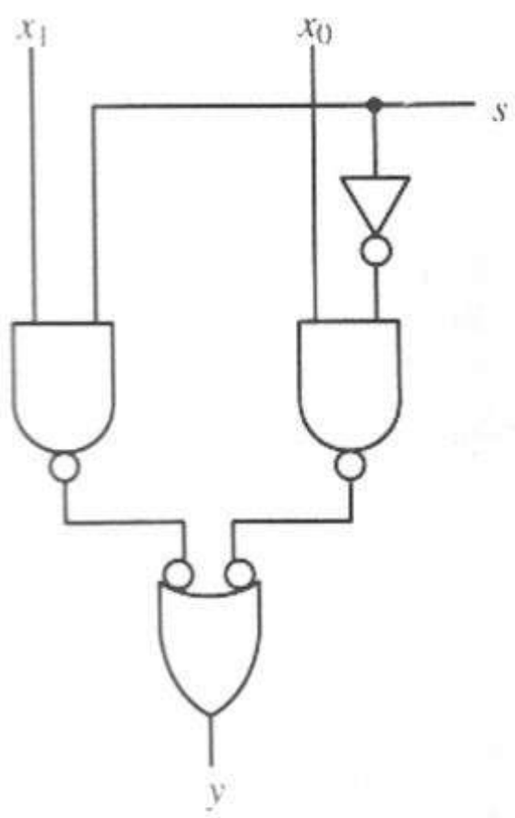


Figura 4.50 Multiplexor 2 a 1.

Ahora analizaremos el diseño de la unidad lógica. Las funciones lógicas de un sistema de cómputo digital son operaciones paralelas que se realizan un bit a la vez. Esto significa que el bit i del resultado, f_{LU_i}, es una función lógica de los bits de entrada a_i y b_i, como lo resume la tabla 4.5. Un método para implantar el módulo LU consiste en utilizar una única compuerta lógica primitiva para realizar cada una de las cuatro funciones lógicas, de modo que la salida de la compuerta deseada sea elegida mediante un multiplexor 4 a 1, según el código de selección S_1S_0. Este circuito aparece en la figura 4.51a, donde las entradas x

TABLA 4.5 FUNCIONES DE LA UNIDAD LÓGICA

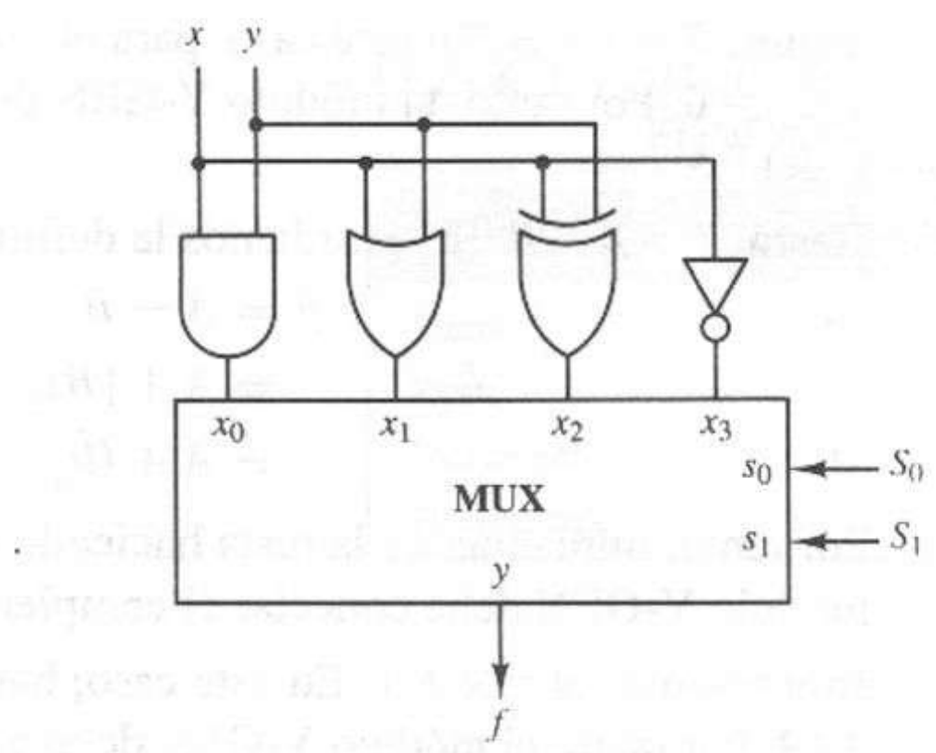
Función		S_1	S_0	f_{LU_i}
AND:	$F = A \cap B$	0	0	$a_i b_i$
OR:	$F = A \cup B$	0	1	$a_i + b_i$
NOT:	$F = \bar{A}$	1	0	$\bar{a}_i$
XOR:	$F = A \oplus B$	1	1	$a_i \oplus b_i$

y y de la LU se conectan a las entradas a_i y b_i de la ALU, respectivamente, y la salida f de la LU se conecta a f_{LU_i}. Podemos realizar el módulo multiplexor 4 a 1 mediante el circuito de la figura 4.23d, descrito anteriormente.

Si es importante la minimización del número de compuertas del módulo LU, podemos graficar la información de la tabla 4.45 en un mapa K, como en las figuras 4.51b y c, obteniendo la siguiente ecuación lógica reducida:

$$f = \bar{S}_1 xy + S_0 x\bar{y} + S_0 \bar{x} y + S_1 \bar{S}_0 \bar{x} \tag{4.48}$$

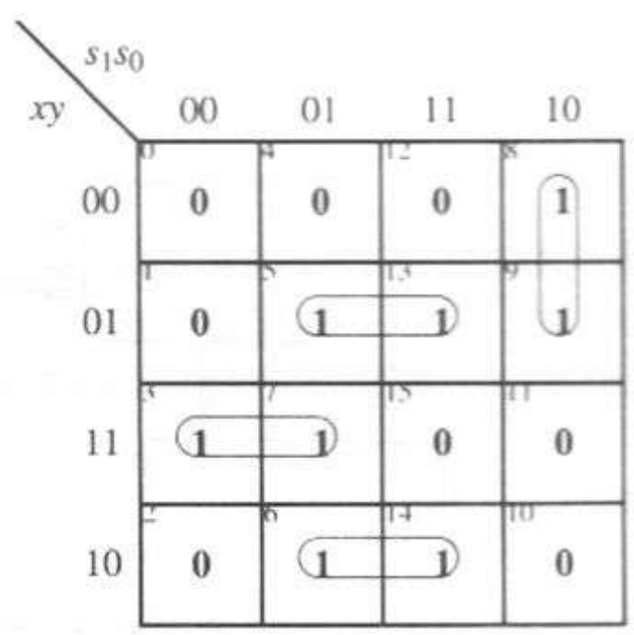
La figura 4.51d muestra una implantación de la ecuación 4.48 mediante compuertas NAND de dos niveles.



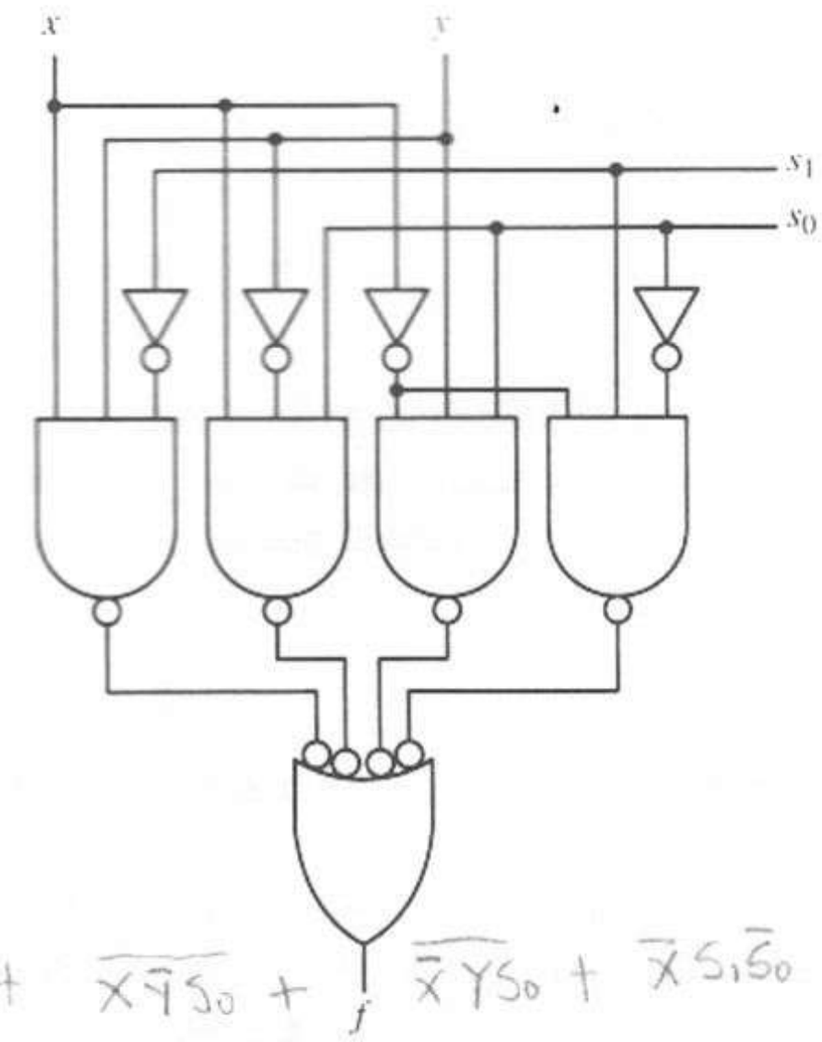
(a)

S_1	S_0	x	y	F_{LU}	
0	0	0	0	0	
0	0	0	1	0	x AND y
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	1	1	x OR y
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	1	
1	0	0	1	1	NOT x
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	1	x XOR y
1	1	1	0	1	
1	1	1	1	0	

(b)



(c)



$\overline{xy\bar{S}_1} + \overline{x\bar{y}S_0} + \overline{\bar{x}yS_0} + \overline{\bar{x}S_1\bar{S}_0}$

(d)

Figura 4.51 Realizaciones del circuito lógico. **(a)** Realización sencilla con multiplexor. **(b)** Tabla de verdad. **(c)** Mapa K de la LU. **(d)** Circuito LU minimizado.

Podemos diseñar la unidad aritmética de nuestra ALU con el método ya descrito en este capítulo. La suma y la resta se realizan con un único circuito sumador completo que usa aritmética de complemento a dos. Para este diseño utilizaremos el circuito sumador completo (FA) de las figuras 4.35f y g. Cuando las secciones de la ALU se conecten en cascada, las etapas FA se conectarán en la configuración del sumador con acarreo en cascada de la figura 4.36. Ahora recordemos que un sumador completo de n bits realiza la expresión

$$F = X + Y + C_{-1} \tag{4.49}$$

donde F, X y Y son números binarios de n bits y C_{-1} es la entrada de acarreo. Podemos implantar con facilidad las cuatro operaciones aritméticas deseadas controlando los valores de Y y C_{-1} en la ecuación 4.49. Por tanto, diseñaremos un circuito que producirá la salida y_i para cada módulo FA, según los bits S_1 y S_0 del código de selección, y otro circuito para obtener C_{-1}. La figura 4.52 muestra la configuración completa de la sección AU de 1 bit. Observe que la entrada x_i del FA simplemente se conecta a la entrada a_i de la ALU.

Consideremos cada una de las cuatro operaciones aritméticas por separado.

Suma: $F = A + B$. En este caso, para el módulo FA, sólo hacemos $X = A$, $Y = B$ y $C_{-1} = 0$. Por tanto, el módulo Y-GEN debe conectar la entrada b_i a la entrada y_i del FA.

Resta: $F = A - B$. Si recordamos la definición del complemento a dos,

$$\begin{aligned} F &= A - B \\ &= A + [B]_2 \\ &= A + (\bar{b}_{n-1} \ldots \bar{b}_1 \bar{b}_0) + 1 \end{aligned} \tag{4.50}$$

Por tanto, implantamos la resta haciendo $y_i = \overline{b_i}$ y $C_{-1} = 1$. En consecuencia, el módulo Y-GEN debe conectar el complemento de b_i a la entrada y_i del FA.

Incremento: $A = A + 1$. En este caso, hacemos $Y = 0$ y $C_{-1} = 1$ en la ecuación 4.49. Por tanto, el módulo Y-GEN debe proporcionar 0 a la entrada y_i de la FA.

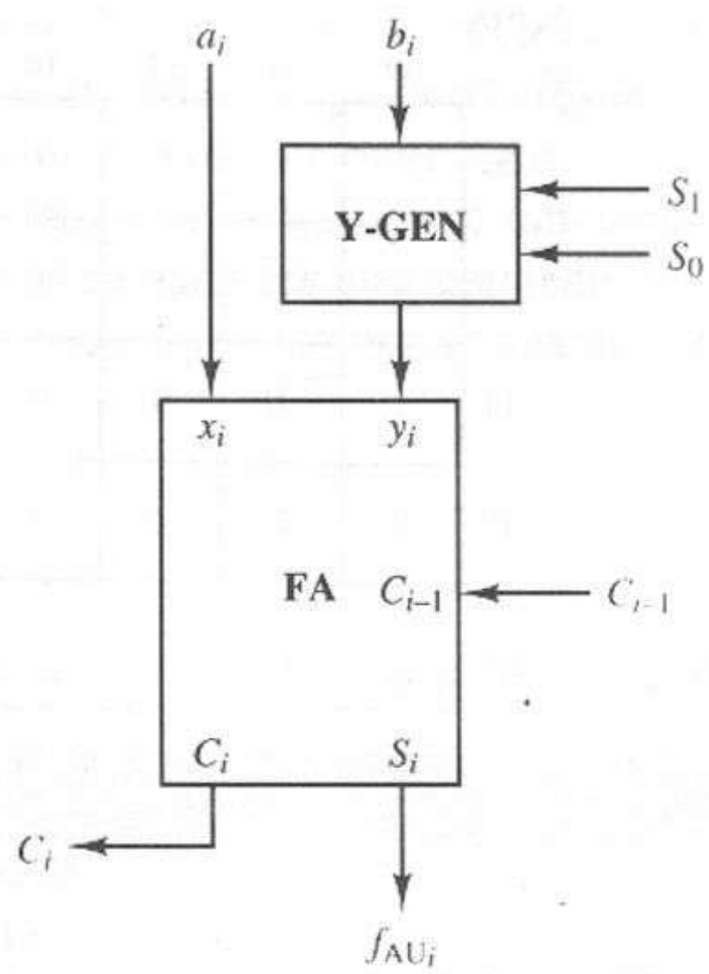


Figura 4.52 Diagrama de bloques de la sección de AU.

Decremento: $A = A - 1$. De nuevo, utilizamos la definición del complemento a dos como sigue:

$$\begin{aligned} F &= A - 1 \\ &= A + (-1) \\ &= A + [00\ldots01]_2 \\ &= A + (11\ldots11) \\ &= A + (11\ldots11) + 0 \end{aligned} \tag{4.51}$$

Por tanto, realizamos la función de decremento haciendo las entradas del FA $y_i = 1$ y $C_{-1} = 0$.

La tabla 4.6 resume el análisis anterior, enumerando los valores necesarios para las entradas y_i y C_{-1} del FA para cada una de las cuatro operaciones aritméticas. Con esta tabla, podemos deducir circuitos lógicos para el módulo Y-GEN de la figura 4.52 y el módulo C-GEN de la figura 4.48b.

TABLA 4.6 VALORES DE y_i Y C_{-1} PARA LAS FUNCIONES ARITMÉTICAS

Función	S_1	S_0	y_i	C_{-1}
Suma	0	0	b_i	0
Resta	0	1	$\bar{b}_i$	1
Incremento	1	0	0	1
Decremento	1	1	1	0

Para el circuito Y-GEN, indicamos y_i sobre un mapa K, como en las figuras 4.53a y b. Obtenemos la ecuación lógica para la salida y_i como sigue:

$$\begin{aligned} y_i &= \bar{S}_1\bar{S}_0 b_i + S_0\bar{b}_i + S_1 S_0 \\ &= \bar{S}_0(\bar{S}_1 b_i) + S_0(S_1 + \bar{b}_i) \\ &= S_0 \oplus (\bar{S}_1 b_i) \end{aligned} \tag{4.52}$$

La figura 4.53c muestra un circuito lógico que realiza la ecuación 4.52.

Para el circuito C-GEN, indicamos C_{-1} sobre un mapa K, como en las figuras 4.54a y b y obtenemos la siguiente ecuación lógica:

$$\begin{aligned} C_{-1} &= S_1\bar{S}_0 + \bar{S}_1 S_0 \\ &= S_1 \oplus S_0 \end{aligned} \tag{4.53}$$

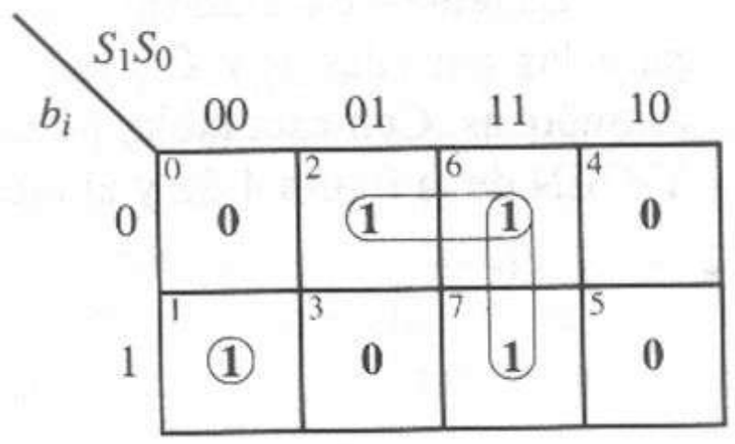
Realizamos la ecuación 4.53 mediante una única compuerta XOR, como en la figura 4.54c.

Ahora, formamos la sección ALU de 1 bit interconectando los módulos individuales antes desarrollados (LU, FA, Y-GEN y MUX). El circuito completo para la sección ALU de 1 bit aparece en la figura 4.55.

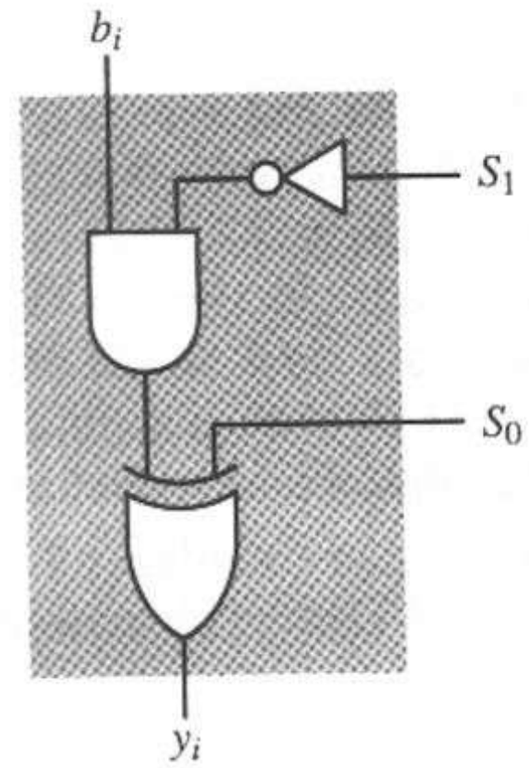
El paso final en el proceso de diseño consiste en crear nuestra ALU de n bits conectando en cascada n secciones ALU de 1 bit y conectando el módulo C-GEN como se muestra en la figura 4.48b.

S_1	S_0	b_i	y_i	
0	0	0	0	Suma
0	0	1	1	
0	1	0	1	Resta
0	1	1	0	
1	0	0	0	Incremento
1	0	1	0	
1	1	0	1	Decremento
1	1	1	1	

(a)



(b)

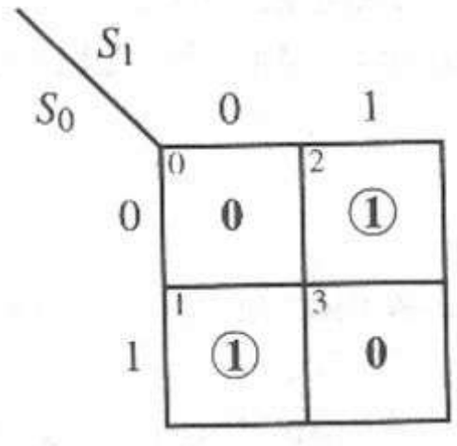


(c)

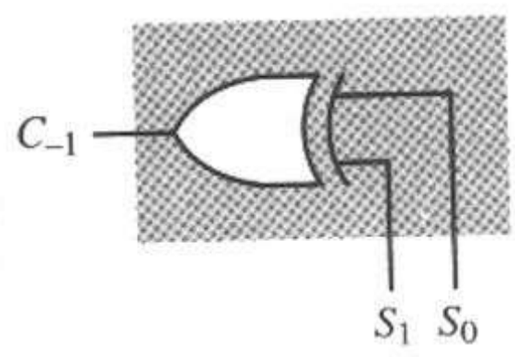
Figura 4.53 Diseño del módulo de circuito Y-GEN. **(a)** Tabla de verdad. **(b)** Mapa K. **(c)** Circuito lógico.

S_1	S_0	C_{-1}	
0	0	0	Suma
0	1	1	Resta
1	0	1	Incremento
1	1	0	Decremento

(a)



(b)



(c)

Figura 4.54 Diseño de módulo de circuito C-GEN. **(a)** Tabla de verdad. **(b)** Mapa K. **(c)** Circuito lógico.

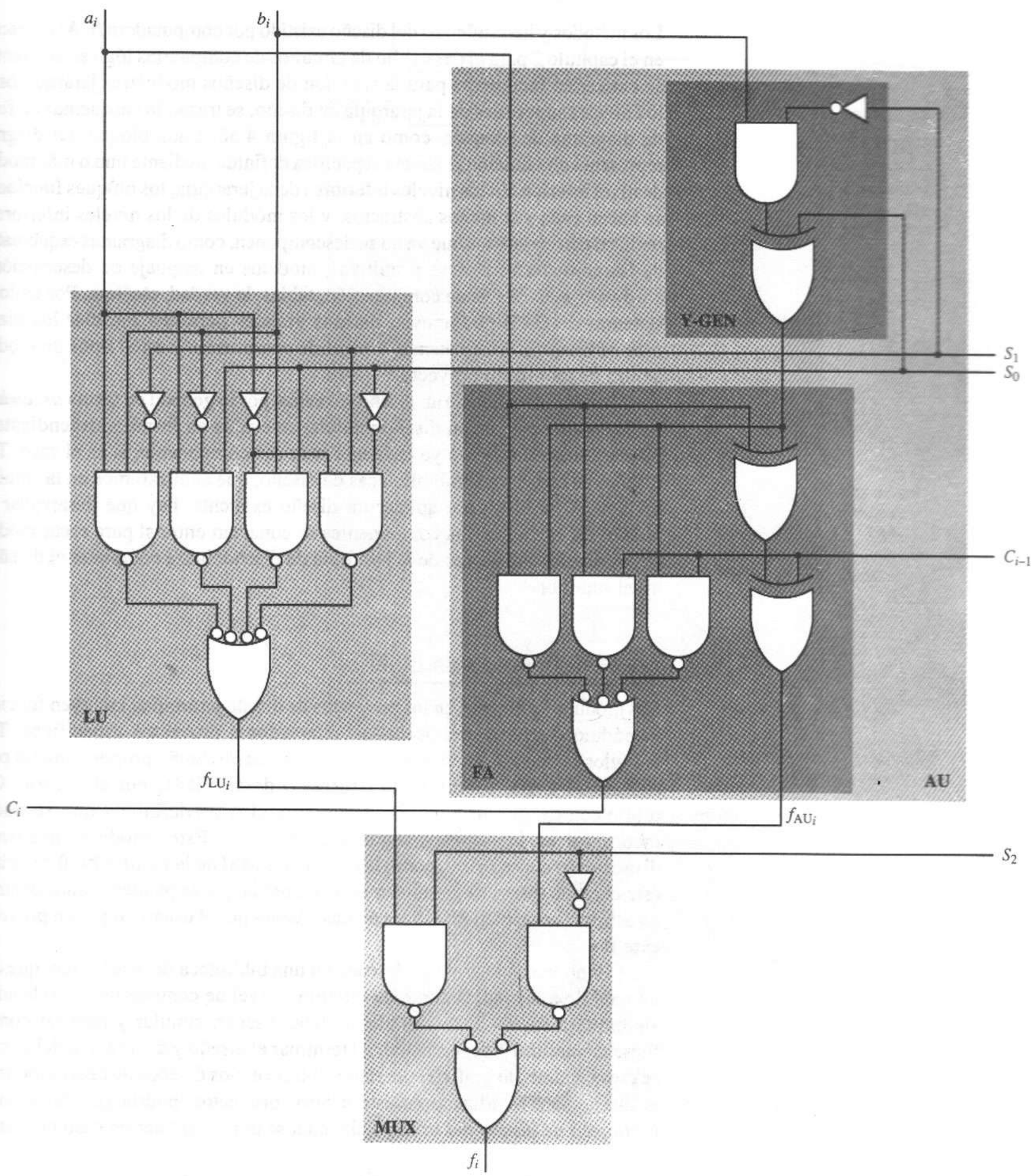


Figura 4.55 Sección de ALU completa de 1 bit.

4.9 Diseño de sistemas modulares asistido por computadora

Los métodos y herramientas del diseño asistido por computadora (CAD) descritos en el capítulo 2 para el desarrollo de circuitos de compuertas lógicas elementales se extienden fácilmente para la creación de diseños modulares jerárquicos. En los niveles superiores de la jerarquía de diseño, se trazan los esquemas en forma de diagrama de bloques, como en la figura 4.56. Cada bloque del diagrama representa una función abstracta específica definida mediante uno o más modelos de nivel inferior. En los niveles inferiores de la jerarquía, los bloques funcionales se hacen cada vez menos abstractos, y los módulos de los niveles inferiores se realizan con elementos que ya no se descomponen, como diagramas esquemáticos de las compuertas lógicas primitivas, modelos en lenguaje de descripción de hardware, ecuaciones de conmutación, tablas de verdad, etcétera. Por tanto, los sistemas de CAD que apoyan diseños grandes permiten mezclar los diagramas de bloques, los esquemas a nivel de compuertas y otros tipos de modelos dentro de un mismo proyecto.

La implantación de un diseño se realiza por lo general en forma *ascendente*. Cada componente de un diseño modular se evalúa en forma independiente. Se puede utilizar un diseño ya existente para ese componente, si es el caso. Tales diseños se conservan en bibliotecas de diseño, que analizaremos en la siguiente sección. Si no se puede aplicar un diseño existente, hay que desarrollar uno nuevo. En este caso, los componentes se conectan entre sí para crear módulos en los niveles superiores de la jerarquía de diseño, hasta completar el diseño de nivel superior.

4.9.1 Bibliotecas de diseño

Los módulos utilizados en los proyectos de diseño jerárquico incluyen funciones y módulos estándar prediseñados, adaptados a proyectos específicos. Tales módulos se conservan en una o más *bibliotecas de diseño* proporcionadas por el sistema de CAD, obtenidas de terceros o desarrolladas por el usuario. Cada biblioteca de diseño contiene varios módulos predefinidos que se pueden incorporar en los diseños según sea necesario. Estos módulos pueden ser dispositivos comerciales, como las funciones MSI de la familia 7400 descrita en este capítulo, funciones estándar de la industria que se pueden incluir dentro de otros circuitos, o bien funciones únicas creadas por el usuario o por un proveedor externo.

Si no existe la función deseada en una biblioteca de diseño, hay que crear un módulo a la medida como un circuito a nivel de compuertas o modelado de alguna otra forma. Este módulo se debe diseñar, simular y depurar como si fuese un circuito independiente. Al terminar el diseño y depuración del circuito, se crea un símbolo gráfico para representarlo en los diseños de nivel superior. Si el diseño será reutilizado en éste u otros proyectos, podría guardarse en una *biblioteca de diseño del usuario*, del cual se podría extraer en caso necesario.

Módulos de biblioteca

Como en el caso de las compuertas lógicas primitivas, cada módulo de nivel superior en una biblioteca de diseño tiene dos componentes: un símbolo gráfico

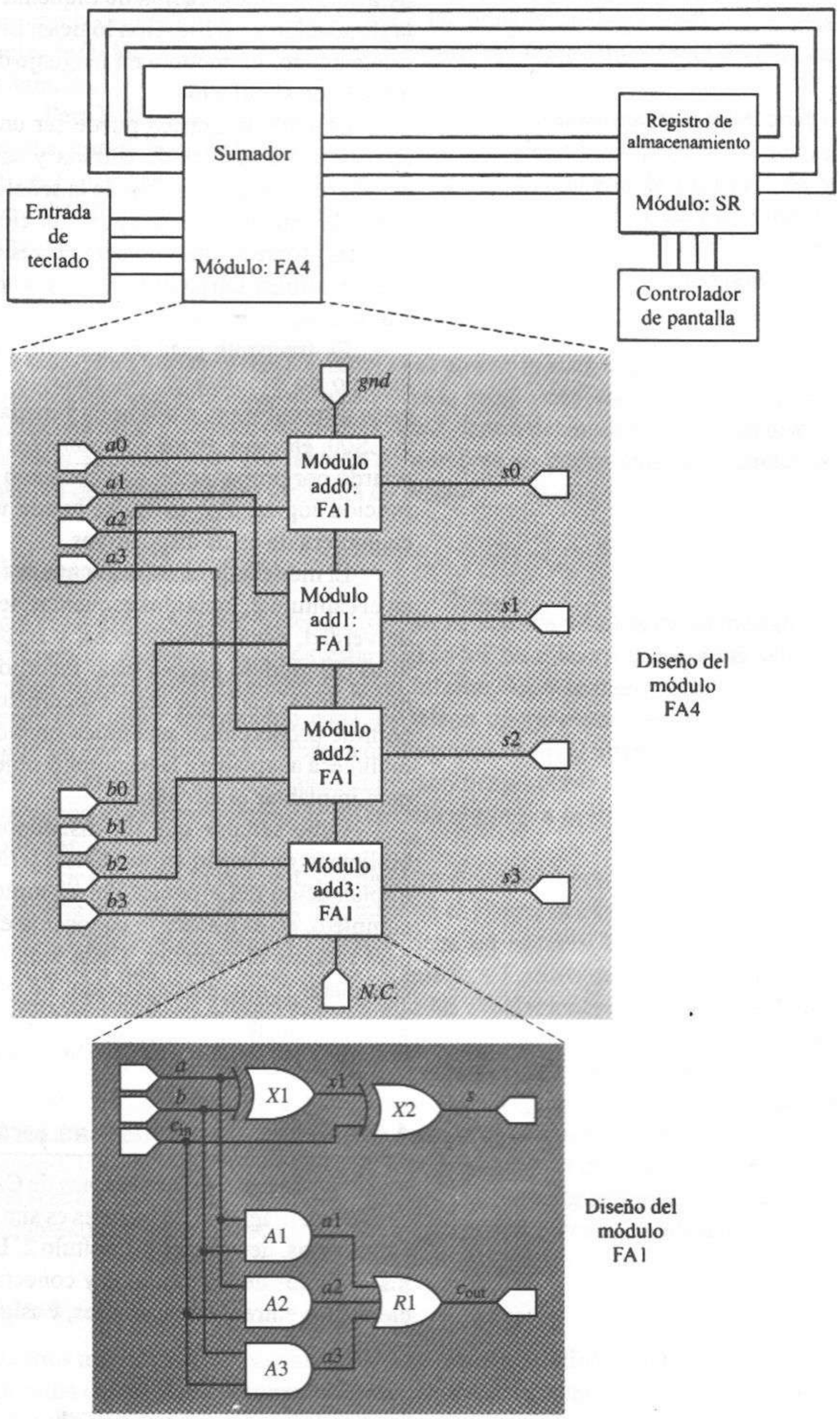


Figura 4.56 Diseño jerárquico.

para su uso en la creación de esquemas y un modelo funcional que se utiliza en la simulación y verificación lógicas. El modelo funcional puede ser un diagrama esquemático, un modelo en lenguaje de descripción de hardware, o algún otro modelo de simulación.

El símbolo gráfico puede ser un bloque sencillo o una forma compleja, con todas las señales de entrada y salida indicadas como *pines*. Cada *pin* se caracteriza por la dirección de la señal correspondiente y el número de bits de la señal. En muchos casos, se trazan ciertas líneas especiales llamadas *whiskers* (bigotes) sobre el símbolo para representar la posición de los *pines*. También se pueden utilizar burbujas de inversión para indicar las entradas o salidas activas bajas.

Es frecuente usar un símbolo gráfico dado para representar más de un modelo de simulación. Por ejemplo, las compuertas NAND de dos entradas están disponibles en una amplia variedad de componentes estándar: 7400, 74S00, 74LS00, etcétera. Estos componentes difieren en algunos parámetros, como el retardo por propagación y el consumo de energía, pero todos realizan la misma función lógica. Por tanto, se puede usar el mismo símbolo para representar cualquiera de estos dispositivos.

El modelo de simulación puede tener cualquiera de los formatos descritos en el capítulo 2, incluidos esquemas, lenguaje de descripción de hardware, tabla de verdad, ecuaciones lógicas, y otros. Los parámetros de tiempo y de otros tipos del módulo pueden estar incorporados al modelo de simulación o dejarse indefinidos, para que sean especificados por el usuario. Este último método permite proporcionar una única función, independiente de la tecnología, en un biblioteca adaptable a la tecnología objetivo que será utilizada en última instancia para implantar el circuito.

Cabe señalar que un diseñador puede incorporar un módulo de una biblioteca de diseño en un diseño jerárquico sin tener acceso a los detalles de implantación del módulo. Por ejemplo, si se va a utilizar un módulo sumador completo, no es necesario conocer la estructura de circuito específica empleada para realizar el sumador. Cada módulo de una biblioteca de diseño se puede considerar como una caja negra. Lo único que se necesita es el símbolo gráfico del módulo, que define sus señales de entrada y salida, una descripción de la función y los parámetros de tiempo y de otros tipos.

4.9.2 Trazo de esquemas jerárquicos

En la mayor parte de los sistemas de CAD, el proceso de trazo de un esquema en el nivel de diagrama de bloques es similar al trazo de un diagrama en el nivel de compuertas, descrito en el capítulo 2. El proceso se inicia eligiendo y colocando los símbolos de los módulos y conectores en la hoja de dibujo, conectando los elementos entre sí con alambres, y asignando nombres simbólicos a los módulos y redes.

Convenciones de nombres

Cada copia de un módulo utilizada en un diagrama se denomina *instancia* de ese módulo. Cada instancia tiene asignado un único nombre de referencia que será utilizado durante la simulación y documentación del diseño. Por ejemplo, en la figura 4.56, el módulo sumador completo 4 a 1 FA4 tiene asignado el nombre de referencia *Adder*. Este módulo se realiza mediante cuatro instancias

del módulo sumador completo de 1 bit FA1. Estas instancias tienen asignados los nombres *add0, add1, add2* y *add3*. De manera análoga, cada compuerta lógica individual y línea de señal del módulo FA1 tiene asignado un nombre de referencia.

Para la depuración, con frecuencia es necesario examinar una señal dentro de una instancia específica de un módulo. Por ejemplo, supongamos que se sospecha del funcionamiento del módulo sumador completo de 1 bit FA1 de la figura 4.56 y que se desea examinar el valor de la señal $x1$ de este circuito durante la simulación de todo el sistema. Como hay cuatro instancias de FA1, el nombre de la instancia se incorpora en el nombre de la señal como sigue:

nombre_del_módulo_de_nivel_superior/nombre_del_módulo_de_nivel2:nombre-de_la_señal

Por tanto, para vigilar la línea de señal $x1$ dentro de la instancia *add2* del sumador completo de la figura 4.56, el nombre de la señal sería *Adder/add2:x1*, lo que indica una señal $x1$ dentro del módulo *add2*, dentro del módulo *Adder* del diseño de nivel superior. Esta convención de nombres se puede ampliar a cualquier número de niveles en una jerarquía de diseño.

Conectores jerárquicos

Para crear esquemas jerárquicos, los *pines* de entrada y salida de un módulo en un nivel de abstracción deben estar asociados de manera única a las señales correspondientes del esquema de nivel inferior. Esto se maneja en el esquema de nivel inferior terminando cada señal con un *conector jerárquico*, lo que significa que la señal será visible en un diagrama de nivel superior. Este conector se asocia al *pin* de entrada o salida correspondiente sobre el símbolo del módulo, asignando el mismo nombre de referencia al *pin* y a la línea de señal unida al conector.

Por ejemplo, la figura 4.57a muestra el símbolo gráfico para un sumador completo de 1 bit y la figura 4.57b muestra el diagrama esquemático correspondiente. Cada *pin* del símbolo gráfico tiene un nombre de referencia correspondiente a una de las líneas de señal que terminan en conectores jerárquicos dentro del esquema. Observe que, como las asociaciones se realizan mediante nombres, las posiciones físicas de los conectores jerárquicos en el esquema no tienen que coincidir con las del *pin* correspondiente en el símbolo.

Muchos sistemas de CAD ofrecen una herramienta para la generación automática de símbolos gráficos a partir de diagramas esquemáticos. Por lo general, los símbolos son similares al que se muestra en la figura 4.57a, es decir, una sencilla figura de bloque con un *pin* por cada conector jerárquico del diagrama esquemático, con los *pines* de entrada a la izquierda y los de salida a la derecha.

Buses

Muchos módulos tienen varias señales de entrada y salida relacionadas entre sí. En estos casos, las interconexiones suelen trazarse como *buses*. Un *bus* es una colección, o haz, de alambres de señal relacionados entre sí. Cada *bus* tiene asignado un nombre simbólico para que se pueda hacer referencia al haz de alambres como una unidad durante la simulación y documentación del diseño de circuito.

Por ejemplo, la figura 4.56 muestra un módulo sumador completo FA4 de 4 bits con ocho entradas y cuatro salidas, que representa tres números de 4 bits.

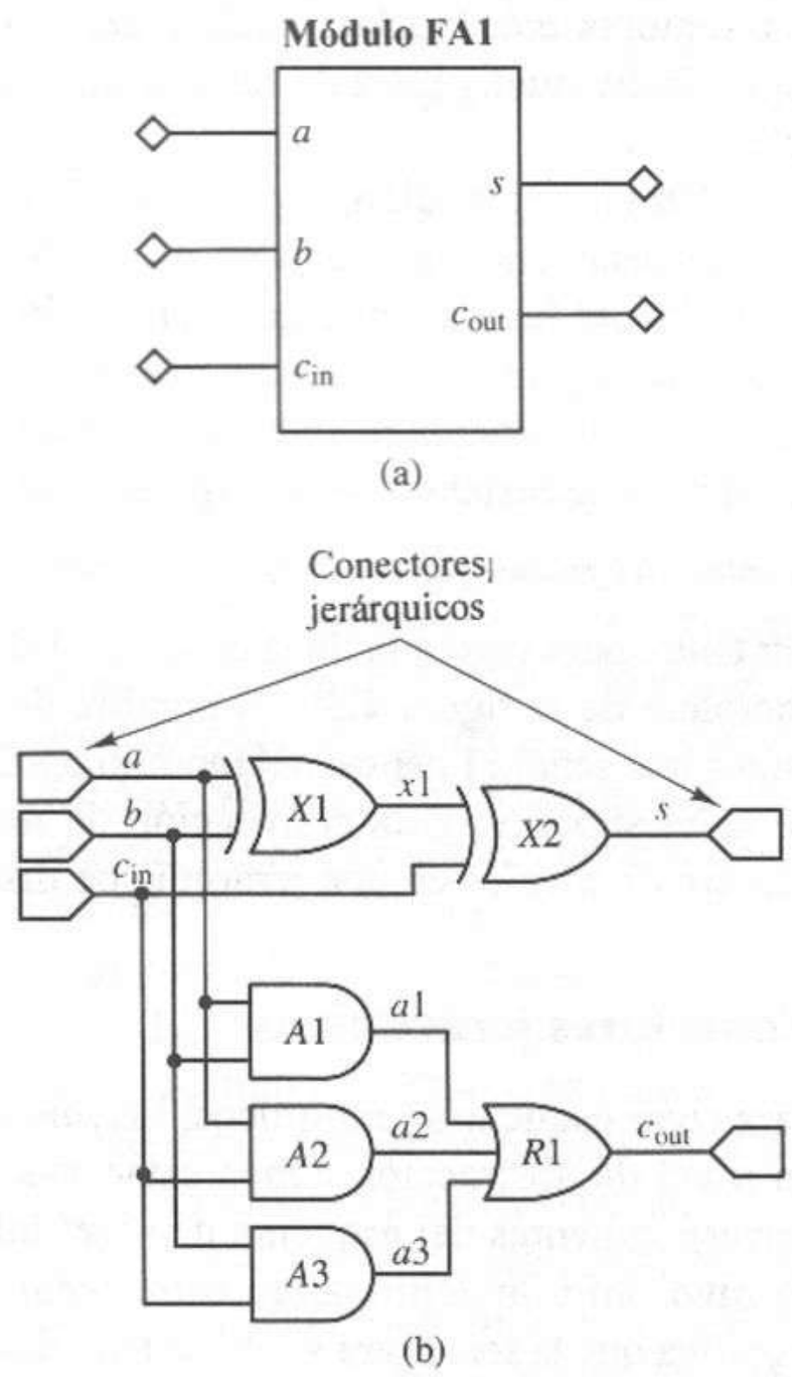


Figura 4.57 Conectores jerárquicos entre un sumador completo (símbolos y esquema). **(a)** Símbolo gráfico. **(b)** Modelo esquemático.

En este caso, sería más conveniente representar los cuatro alambres que transportan cada uno de los tres números como un *bus* de 4 bits, definido como sigue.

$$A(3:0) = \{A(3), A(2), A(1), A(0)\}$$
$$B(3:0) = \{B(3), B(2), B(1), B(0)\}$$
$$S(3:0) = \{S(3), S(2), S(1), S(0)\}$$

Cada alambre individual conserva un nombre único [por ejemplo, $A(3)$] para que pueda conectarse a un *pin* de dispositivo o seleccionarse para estimularlo o vigilarlo durante la simulación lógica. El diagrama de bloques simplificado, con $A(3:0)$, $B(3:0)$ y $S(3:0)$ representados como *buses*, aparece en la figura 4.58a. Esta simplificación es aún más pronunciada en los sistemas de cómputo modernos, en los que se utilizan con frecuencia *buses* de 32 a 64 bits para conectar los componentes entre sí.

En la mayor parte de los programas para captura de esquemas, los *buses* se trazan de la misma forma, como alambres individuales. En algunos casos, se dibuja un alambre normal y después se indica que es un *bus* asignándole un nombre de señal indexado, como $A(3:0)$. En otros casos, se proporciona una herramienta *bus* para trazar el haz de alambres, de modo que el *bus* se indique mediante una línea más gruesa o un color distinto, para distinguirlo de los alambres individuales. Ilustramos el primer caso en la figura 4.58a.

La figura 4.58a también muestra el uso de un *bus extendido*, que es un mecanismo para la separación de alambres individuales o grupos de alambres

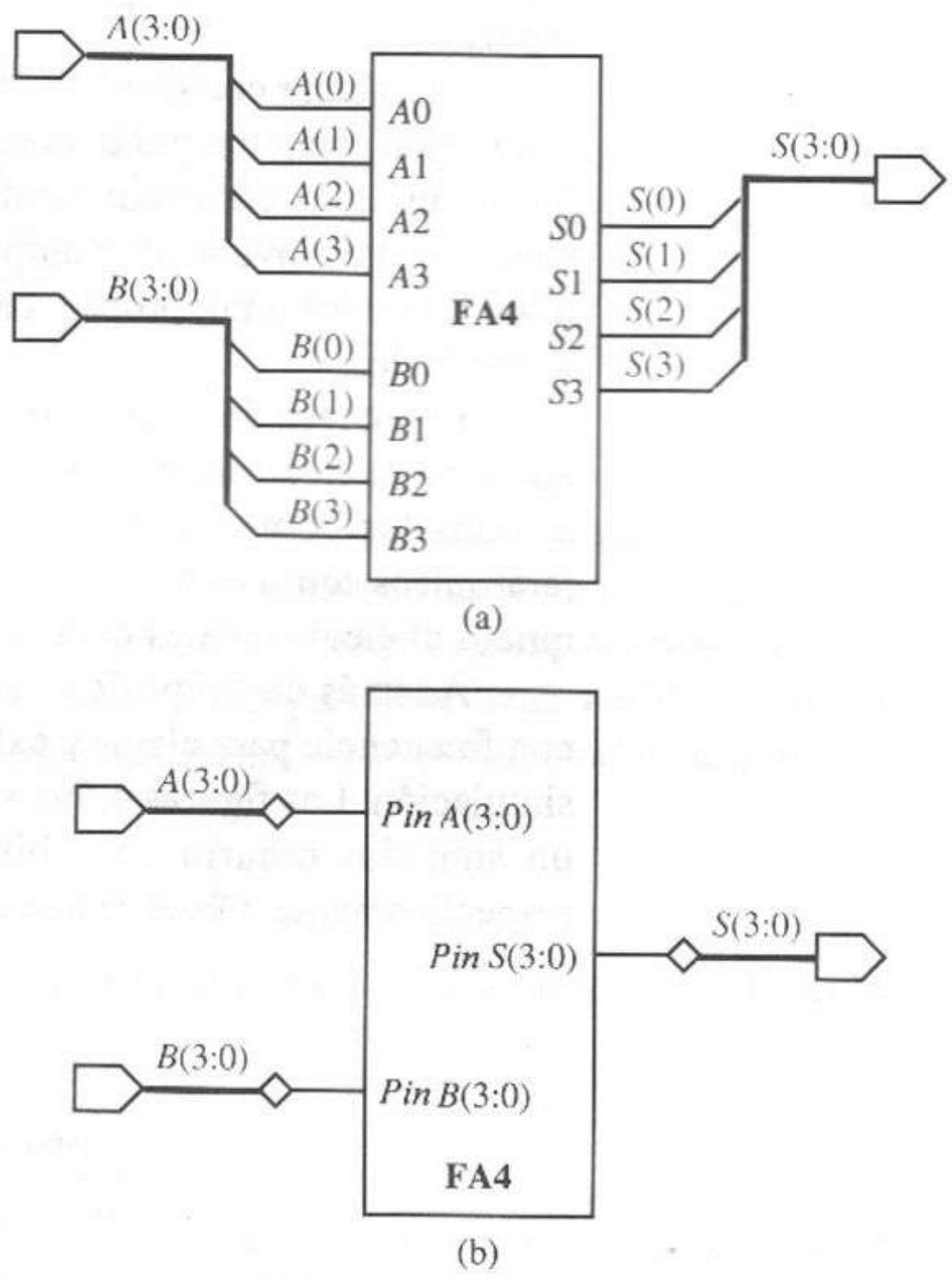


Figura 4.58 Uso de *buses* para simplificar los diagramas lógicos. **(a)** *Bus* con *bus extendido*. **(b)** Módulo con *pines* de varios bits.

de un *bus* para realizar conexiones con *pines* de dispositivos u otros conectores. En esta figura, cada uno de los tres *buses* se separa en alambres individuales para su conexión con los 12 *pines* del símbolo FA4.

Se pueden simplificar aún más los diagramas esquemáticos modulares, definiendo *pines* de varios bits en los módulos, donde cada *pin* representa una colección de conexiones de señal relacionadas entre sí, de la misma forma en que un *bus* representa una colección de alambres relacionados entre sí. Esto permite conectar un *bus* de manera directa con los *pines* de un módulo sin tener que separarlo en alambres individuales. Por ejemplo, la figura 4.58b presenta otra versión del circuito de la figura 4.58a, pero en el que cada *pin* en el módulo representa cuatro conexiones de señal.

4.10 Simulación de sistemas jerárquicos

La verificación y depuración de un sistema suelen realizarse con un método *ascendente*. Primero se simulan y evalúan los componentes inferiores en una jerarquía de diseño usando los métodos descritos en el capítulo 2. Cuando se ha verificado por completo la operación correcta de estos módulos, se pueden verificar los circuitos del siguiente nivel de abstracción que utilizan estos compo-

nentes. Un diseño que no ha sido depurado por completo como circuito independiente es más difícil de depurar dentro del contexto de un diseño de nivel superior.

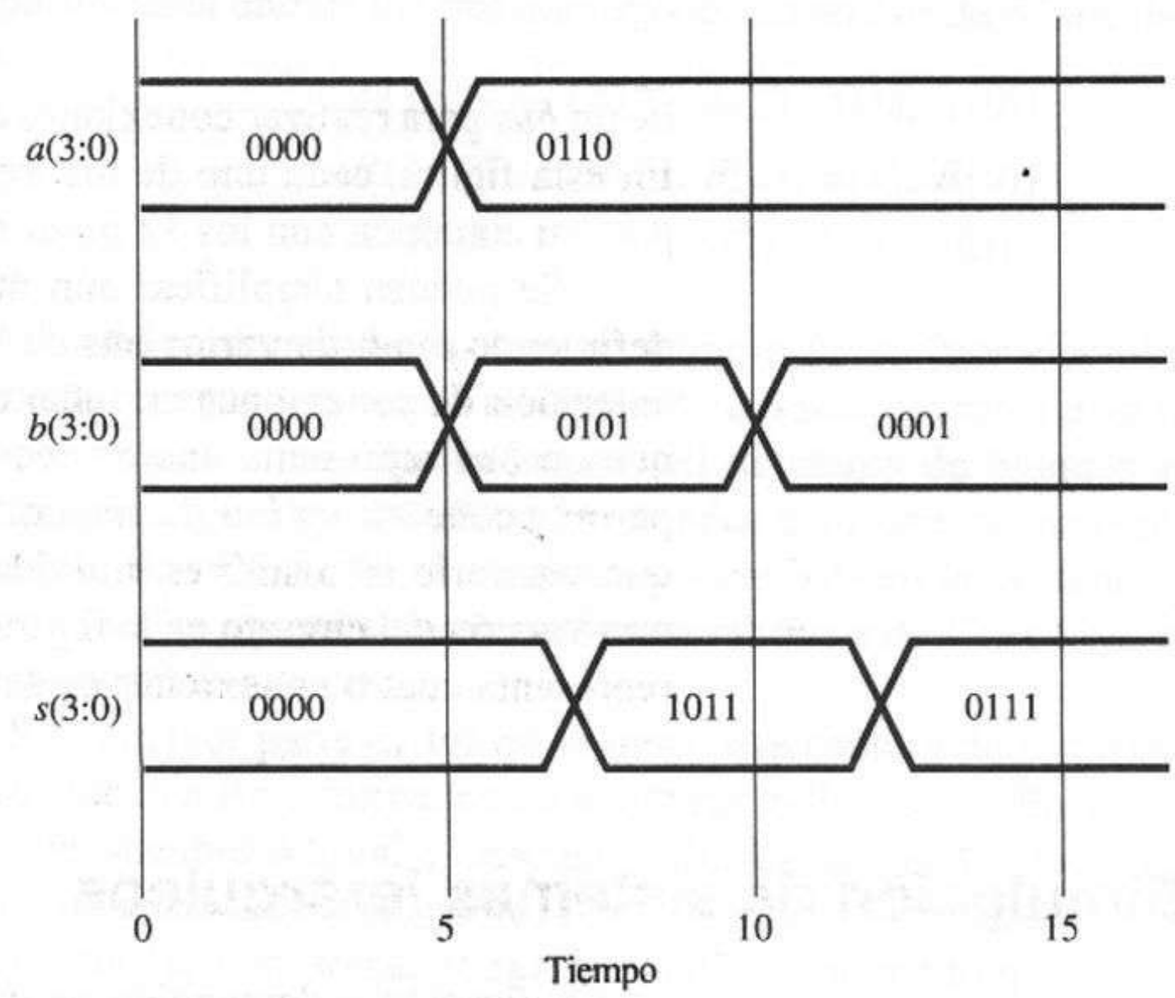
Al trabajar con diseños jerárquicos, la mayor parte de las herramientas de simulación comienza por *aplanar* el circuito; es decir, reemplazando cada módulo por su circuito correspondiente de nivel inferior, hasta que todo el circuito contiene exclusivamente compuertas lógicas primitivas. Entonces se obtienen todos los retardos por propagación y demás parámetros a partir de las compuertas individuales.

El proceso de trabajo con las señales en un diseño jerárquico es similar al que se sigue en un diseño plano. Sin embargo, durante el proceso de aplanamiento, a todos los componentes y alambres de señal se les asignan sus nombres jerárquicos completos, como ya hemos descrito. Utilizamos estos nombres jerárquicos al elegir señales para la simulación y/o vigilancia.

Además de simplificar los diagramas esquemáticos, los *buses* se utilizan con frecuencia para elegir y exhibir las señales de manera más sencilla durante simulación. Las figuras 4.59a y b muestran los resultados de la simulación para un sumador binario de 4 bits, en formatos tabular y de forma de onda, respectivamente. Observe que en la forma de onda cada *bus* se muestra como un

Tiempo	$a(3:0)$	$b(3:0)$	$s(3:0)$
0	0000	0000	0000
5	0110	0101	0000
7	0110	0101	1011
10	0110	0001	1011
12	0110	0001	0111

(a)



(b)

Figura 4.59 Valores de simulación para *buses*. **(a)** Formato tabular. **(b)** Formato de forma de onda.

par de líneas en los estados lógicos 0 y 1, mostrando entre ellas los valores individuales. Hacemos esto porque algunas de las líneas del *bus* podrían ser 1 y otras 0. El cruce de las líneas de señal indica que una o más señales dentro del *bus* han cambiado.

Muchos simuladores manejan mezclas de diagramas esquemáticos, modelos en lenguaje de descripción de hardware, y otros métodos de modelado. En estos casos, la simulación y vigilancia de las líneas de señal se sigue realizando de la manera ya descrita. Sin embargo, se cuenta con diferentes herramientas para depuración de diseño para cada tipo de modelo.

4.11 Resumen

En este capítulo hemos analizado el tema de los circuitos lógicos combinatorios modulares. Hemos intentado abarcar los temas importantes implicados en el análisis y la síntesis de estos circuitos. Además, presentamos y analizamos varios circuitos especiales que se utilizan ampliamente en todas las facetas de la organización y diseño de una computadora digital.

REFERENCIAS BIBLIOGRÁFICAS

1. *The TTL Data Book, Volumen 2.* Dallas, Texas, Texas Instruments, Inc., 1988.

PROBLEMAS

4.1 Deduzca expresiones de conmutación para las salidas 5 y 11 del módulo decodificador 74154. Utilice estas expresiones para describir el funcionamiento del decodificador y la función de las entradas de activación.

4.2 Diseñe un decodificador 4 a 16 usando compuertas lógicas. Las entradas codificadas son $\{D, C, B, A\}$ y las salidas son activas bajas: $\{\overline{O}_0, \overline{O}_1, \ldots, \overline{O}_{15}\}$. El decodificador debe tener una línea de habilitación activa alta E.

4.3 Diseñe un decodificador 5 a 32 con sólo módulos decodificadores 3 a 8. Suponga que cada decodificador 3 a 8 tiene una entrada de habilitación activa baja $\overline{E_1}$, y una entrada de habilitación activa alta E_2.

4.4 Realice cada uno de los siguientes conjuntos de funciones con un único módulo decodificador 74154 y compuertas lógicas de salida (elija compuertas NAND y AND para minimizar el *fan-in* de las compuertas de salida).

(a)
$$f_1(a, b, c, d) = \sum m(2, 4, 10, 11, 12, 13)$$
$$f_2(a, b, c, d) = \prod M(0 \text{ a } 3, 6 \text{ a } 9, 12, 14, 15)$$
$$f_3(a, b, c, d) = \bar{b}c + \bar{a}\bar{b}d$$

(b)
$$f_1(a, b, c, d) = \sum m(0, 1, 7, 13)$$
$$f_2(a, b, c, d) = ab\bar{c} + acd$$
$$f_3(a, b, c, d) = \prod M(0, 1, 2, 5, 6, 7, 8, 9, 11, 12, 15)$$

(c) Repita la parte (a) para los complementos de las tres funciones.

(d) Repita la parte (b) para los complementos de las tres funciones.

4.5 Dado el circuito de la figura P4.5, donde el decodificador tiene salidas activas bajas como se muestra, determine la expresión de conmutación mínima para $f(W, X, Y, Z)$ en forma SOP.

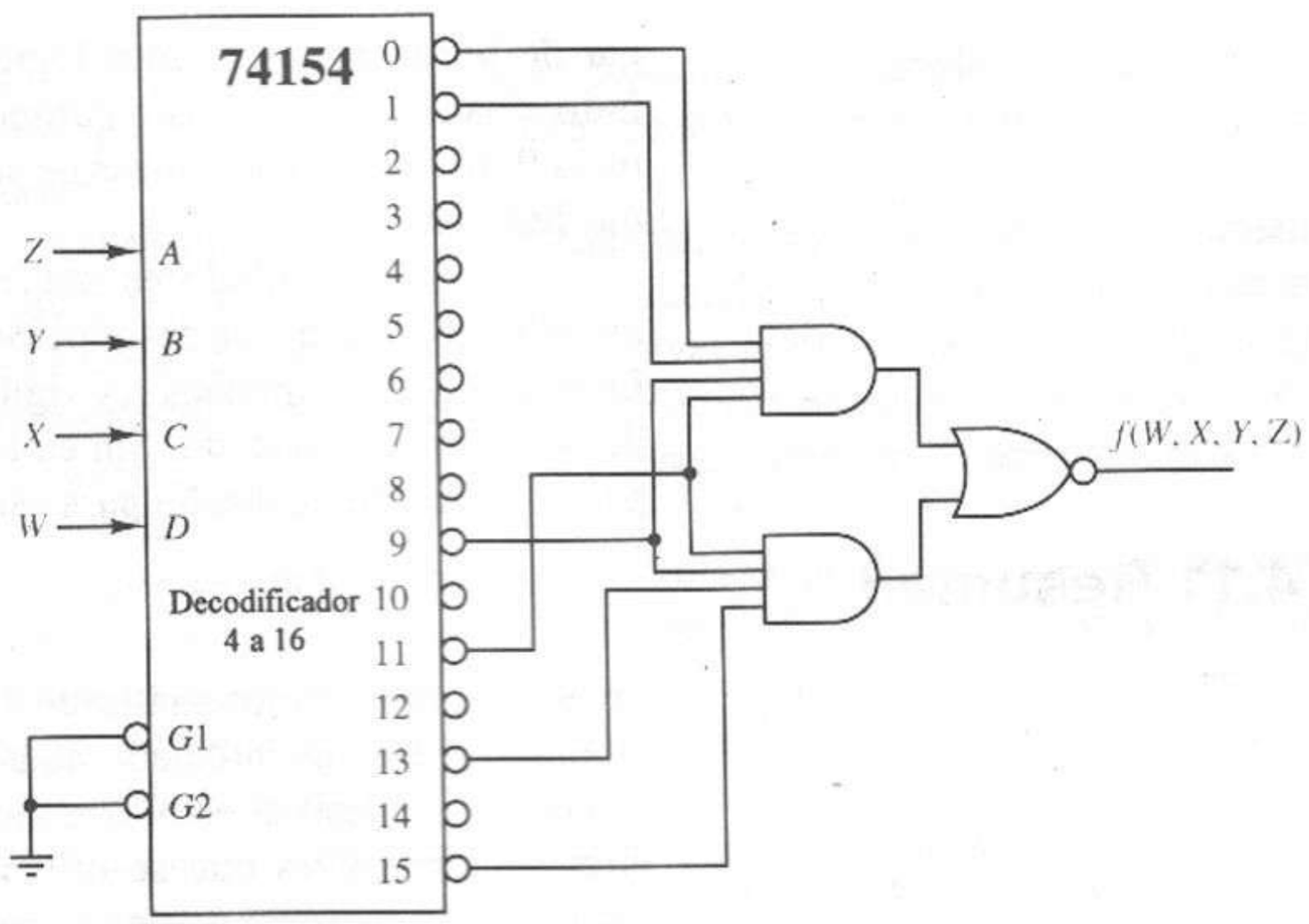


Figura P4.5

4.6 Diseñe un decodificador binario a decimal cuyas entradas sean activas altas, dadas por números codificados en BCD de 4 bits ($x_3x_2x_1x_0$) y cuyas salidas sean las líneas activas bajas (d_9, d_8,..., d_1, d_0). El circuito debe estar minimizado.

4.7 Diseñe un convertidor de código cuya entrada sea un código de 4 bits ($c_3c_2c_1c_0$) que represente los dígitos hexadecimales {0 a 9,A,b,C,d,E,F}, de modo que las salidas controlen un dígito de un *display* de siete segmentos y exhiban el carácter correspondiente. (Las letras *B* y *D* generalmente se exhiben en minúsculas para distinguirlas de los dígitos 8 y 0, respectivamente.)

4.8 Diseñe un circuito lógico que convierta un número de 4 bits del formato de magnitud y signo al formato de complemento a dos. Utilice un circuito AND-OR de dos niveles para cada una de las cuatro salidas.

4.9 Diseñe un convertidor de código que convierta un número de 4 bits del código Gray al código binario.

4.10 Diseñe un circuito codificador 4 a 2 con prioridad que sólo utilice compuertas NOR. Las entradas son $a_3a_2a_1a_0$, donde a_3 tiene la máxima prioridad y a_0 la mínima. Las salidas son y_1y_0, que indican la entrada activa con mayor prioridad, y *G*, la cual indica que al menos una entrada está activa.

4.11 El codificador 74147 de diez líneas con prioridad tiene entradas y salidas activas bajas. Determine la salida del módulo, *DCBA*, para las siguientes combinaciones de entradas.

(a) $(0, 1, \ldots, 9) = (1, 0, 0, 0, 0, 0, 1, 1, 1, 1)$

(b) $(0, 1, \ldots, 9) = (1, 0, 0, 0, 1, 0, 0, 0, 1, 0)$

4.12 Deduzca expresiones de conmutación para las salidas de las compuertas AND controladas por las entradas D_3 y D_6 del módulo multiplexor 74151A. Utilice estas expresiones para describir el funcionamiento del multiplexor y la función de la entrada de estroboscopio (activación).

4.13 Diseñe un circuito multiplexor 5 a 1 minimizado hasta donde sea posible.

4.14 Diseñe un multiplexor de tres entradas y 3 bits. Utilice sólo compuertas NAND.

4.15 Diseñe un multiplexor 8 a 1 sólo con módulos multiplexores 4 a 1, sin líneas de activación. (No utilice compuertas adicionales.)

4.16 Diseñe un multiplexor 32 a 1, utilizando:

(a) Sólo módulos 74151A. (No utilice compuertas adicionales.)

(b) Dos módulos 74150 y un multiplexor 4 a 1.

(c) Dos módulos 74150, un inversor y una compuerta NAND.

4.17 Diseñe un multiplexor dual (2 bits) de 16 entradas con sólo módulos 74151A, compuertas OR e inversores.

4.18 Realice las siguientes funciones con un módulo multiplexor 4 a 1.

(a) $f_1(a, b, c) = \sum m(2, 4, 5, 7)$

(b) $f_2(a, b, c) = \prod M(0, 6, 7)$

(c) $f_3(a, b, c) = (a + \bar{b})(\bar{b} + c)$

4.19 Realice las siguientes funciones con un módulo multiplexor 74151A.

(a) $f(b, c, d) = \sum m(0, 2, 3, 5, 7)$

(b) $f(b, c, d) = \bar{c} + b$

(c) $f(a, b, c, d) = \prod M(0, 1, 2, 3, 6, 7, 8, 9, 12, 14, 15)$

4.20 Determine la lista de mintérminos de la función $f(A, B, C, D)$ realizada por el circuito de la figura P4.20.

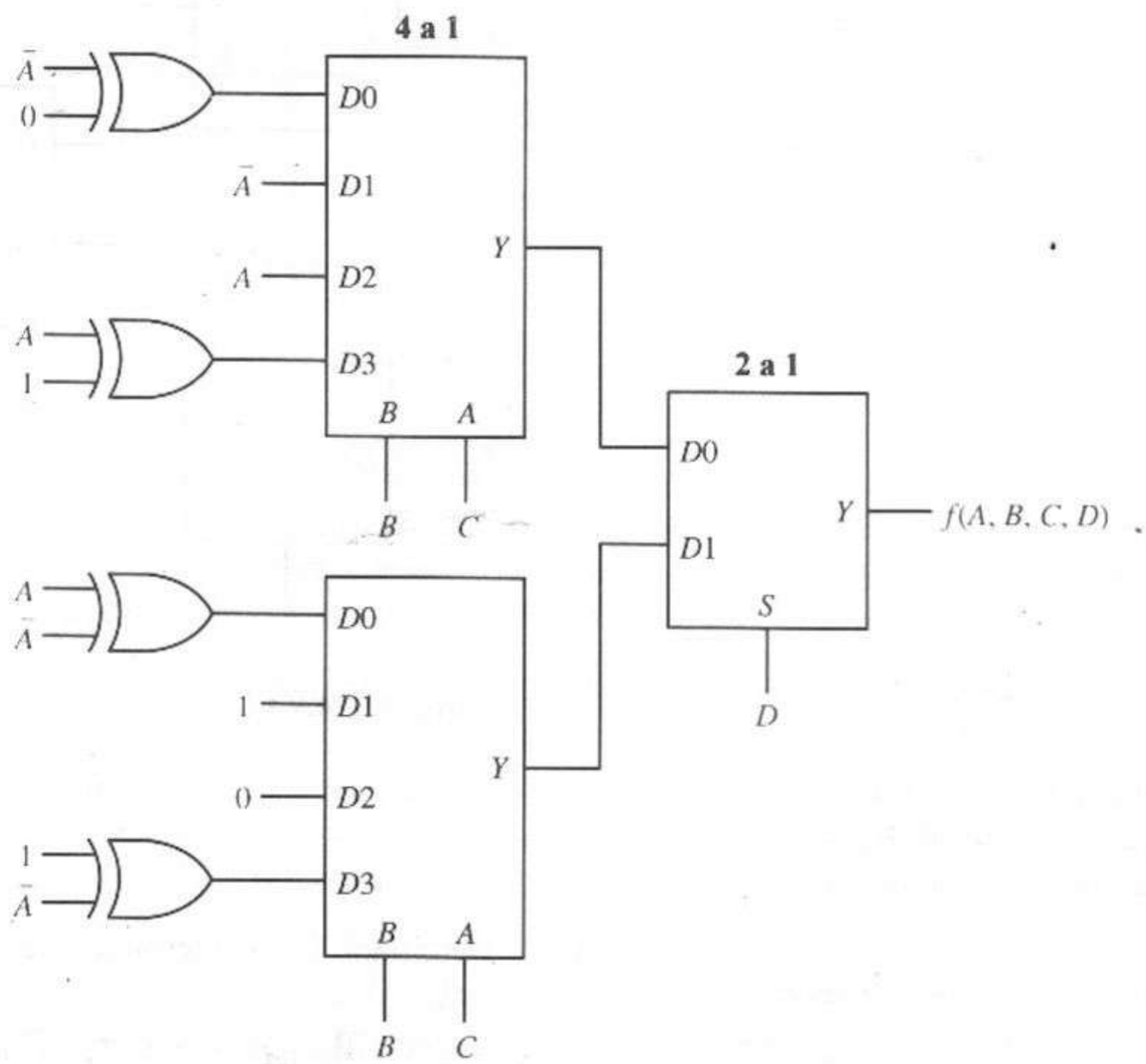


Figura P4.20

4.21 Para el circuito de la figura P4.21, que sólo contiene multiplexores 2 a 1 (con todas las entradas y salidas activas altas), determine los mintérminos de la función de salida $f(A,B,C,D)$.

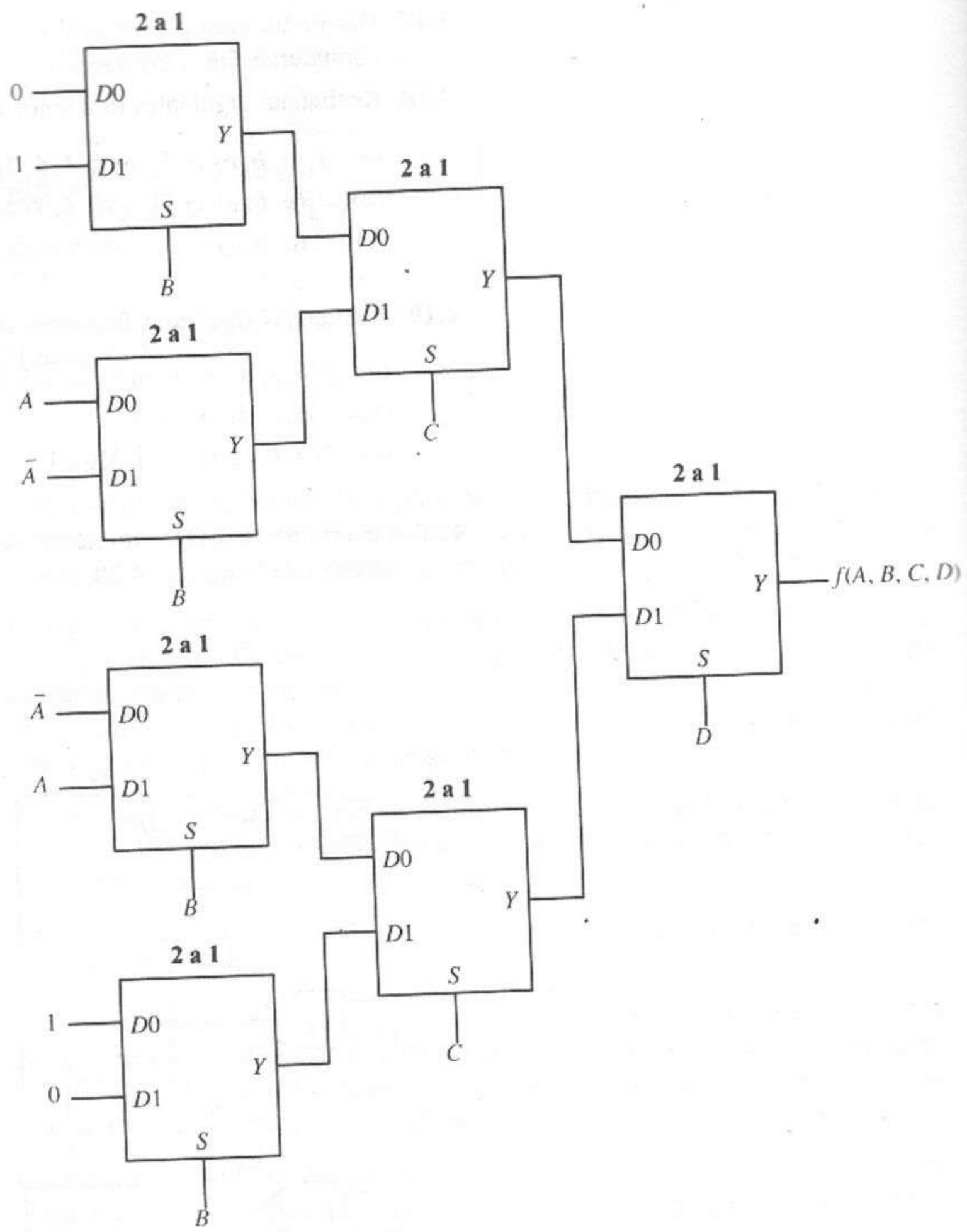


Figura P4.21

4.22 Determine los mintérminos de la función realizada por el circuito de la figura P4.22.

4.23 Sea la función $f(Q, R, S, T) = \sum m(4, 5, 6, 7, 8, 13, 14, 15)$. Utilice el circuito dado en la figura P4.23 para implantar la función conectando en forma adecuada las entradas Q, R, S, T a las compuertas NAND y al multiplexor 4 a 1, conectando

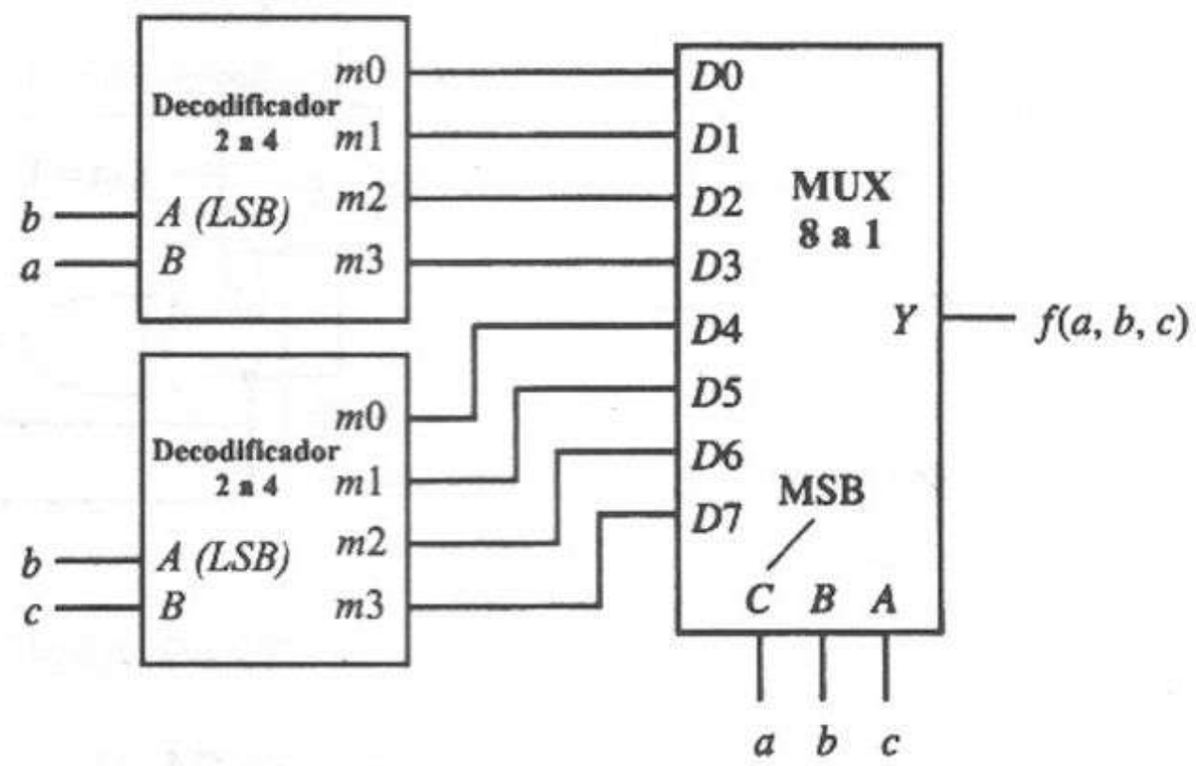


Figura P4.22

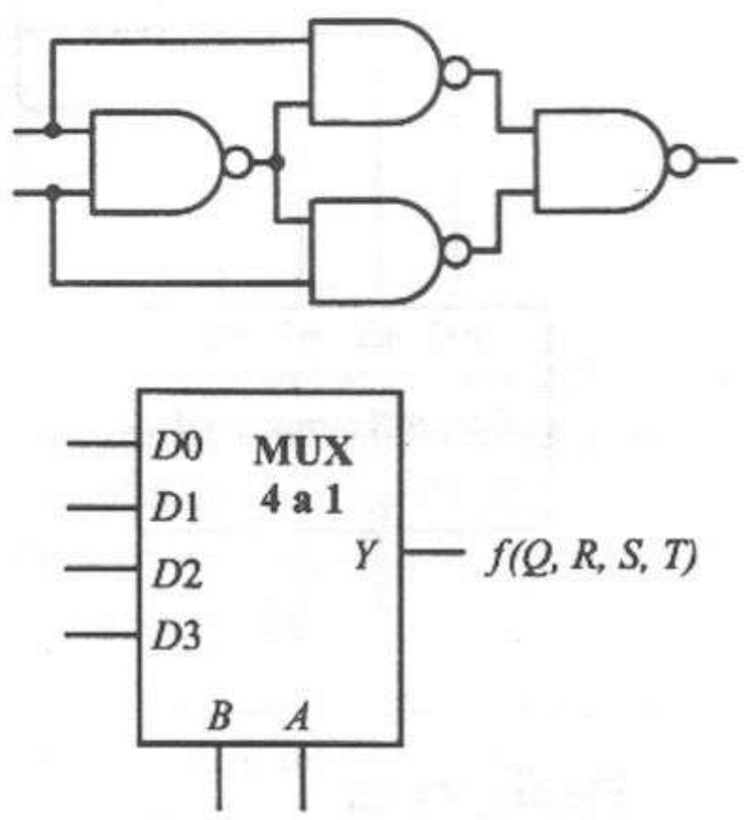


Figura P4.23

también la salida de la compuerta NAND a la(s) entrada(s) adecuada(s) del multiplexor. Las únicas entradas disponibles son Q, R, S, T; no se dispone de 0, 1, $\overline{Q}$, $\overline{R}$, $\overline{S}$, ni $\overline{T}$. (Se requieren nueve conexiones: tres para el circuito NAND y seis para el MUX.) Considere B como el MSB para el multiplexor.

4.24 Determine la función realizada por el circuito de la figura P4.24 en forma de lista de mintérminos. Considere que B es el MSB del multiplexor.

4.25 Dado el circuito de la figura P4.25, proporcione el resultado $f(a, b, c, d)$ en forma de lista de mintérminos. Considere que B es el MSB del decodificador y del multiplexor. Suponga que se utiliza lógica positiva (entradas y salidas activas altas).

4.26 Diseñe un módulo sumador completo con entradas de datos A y B, acarreo de entrada C_{in}, salida de suma S y acarreo de salida C_{out}.

(a) Utilice un decodificador 3 a 8 y compuertas NAND

(b) Utilice un multiplexor de 4 entradas y 2 bits

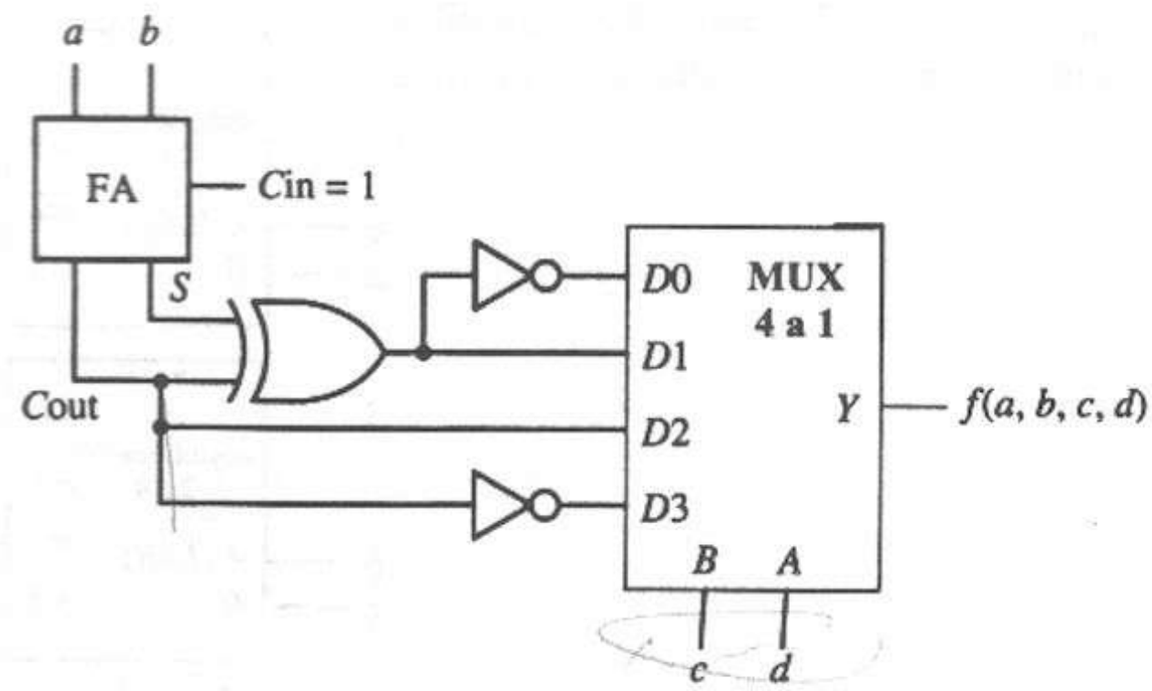


Figura P4.24

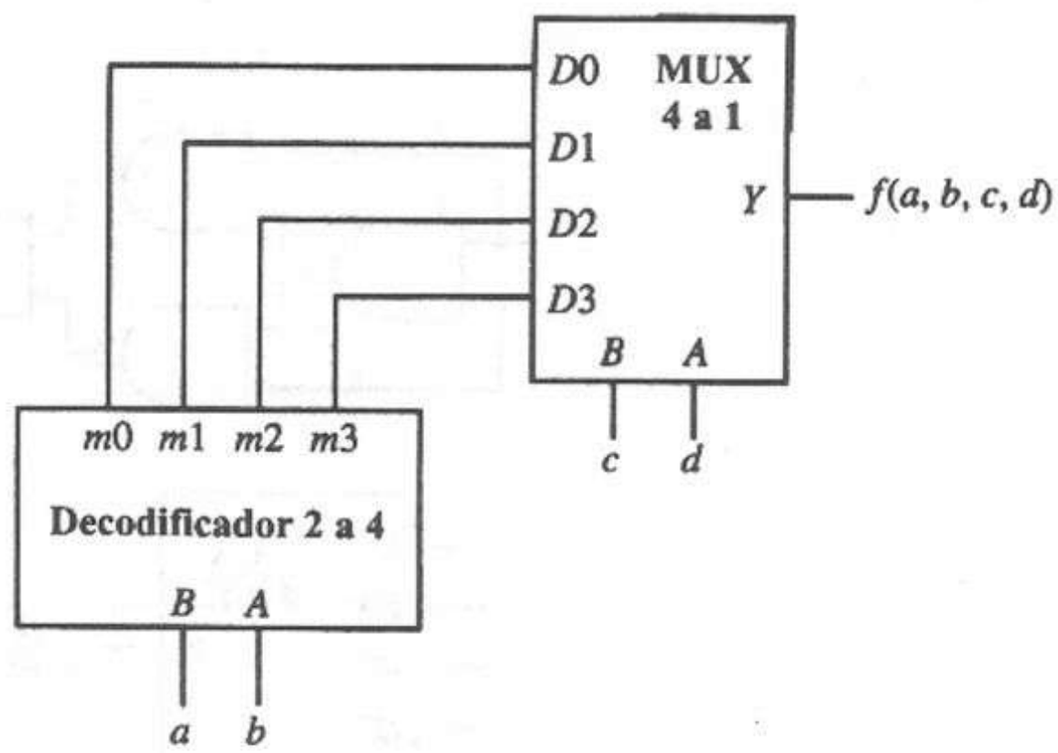


Figura P4.25

4.27 Tres sensores de temperatura, que se muestran en la figura P4.27, indican la temperatura medida con valores binarios de 8 bits en sus líneas de salida T_7 a T_0. Muestre mediante un diagrama de bloques la forma de utilizar módulos multiplexores para que un microprocesador de 8 bits lea cualquiera de estos sensores utilizando sus líneas de entrada de datos, D_7 a D_0, y emitiendo sólo una dirección de 2 bits, A_1A_0.

4.28 Muestre la forma de obtener resultados con una codificación correcta para la operación $A + B$, donde A y B están codificados en el sistema numérico de complemento a dos (valores con n bits), cuando

(a) $A \geq 0$ y $B \geq 0$

(b) $A \geq 0$ y $B < 0$

(c) $A < 0$ y $B < 0$

4.29 Diseñe un circuito sumador de 2 bits mediante un circuito de compuertas NAND de dos niveles para cada salida. Las entradas son los números binarios de 2 bits a_1a_0 y b_1b_0. Las salidas son la suma binaria de 2 bits s_1s_0 y el acarreo de salida c_1.

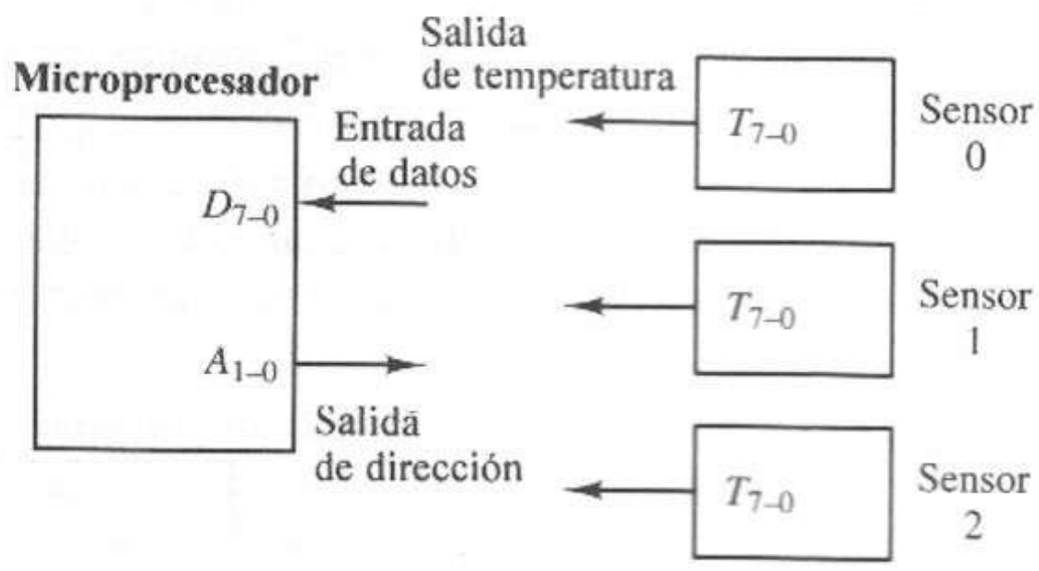


Figura P4.27

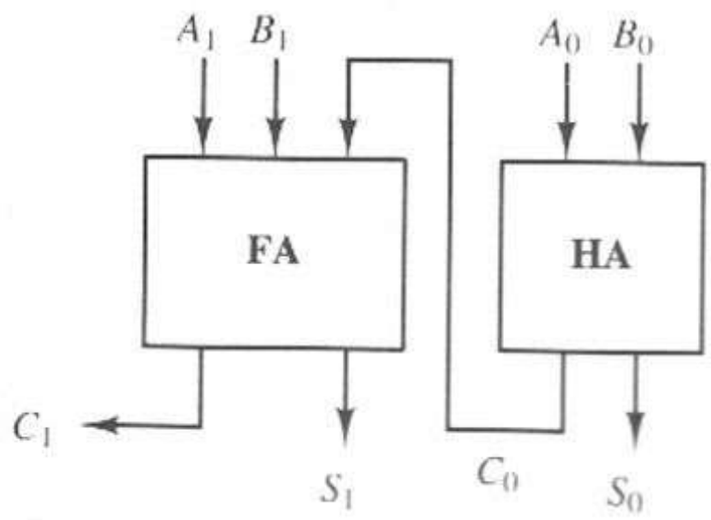


Figura P4.30

4.30 En el diseño del problema 4.29, muestre que el circuito resultante es de hecho una cascada de un medio-sumador de 1 bit y un sumador completo de 1 bit, como se muestra en la figura P4.30.

4.31 Diseñe un sumador con acarreo en cascada de 16 bits, utilizando sólo módulos sumadores 7483.

4.32 El circuito de la figura P4.32, compuesto por cinco medios-sumadores, suma 4 bits, como lo muestra la tabla de verdad. ¿Qué función aparece en las salidas que tienen signos de interrogación?

4.33 Utilice únicamente medios-sumadores para obtener un circuito que sume 3 bits, x_i, y_i, z_i, produciendo los bits de acarreo y de suma c_i, s_i, como se muestra en la tabla siguiente.

x_i	y_i	z_i	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

4.34 Diseñe un sumador completo de 3 bits con acarreo en prospectiva, en vez de acarreo en cascada.

4.35 Diseñe un módulo restador completo de 1 bit, con sólo compuertas NOR, y después construya un restador de 4 bits que sólo utilice estos módulos.

4.36 Describa la condición de desbordamiento aplicada a la suma y resta en complemento a dos.

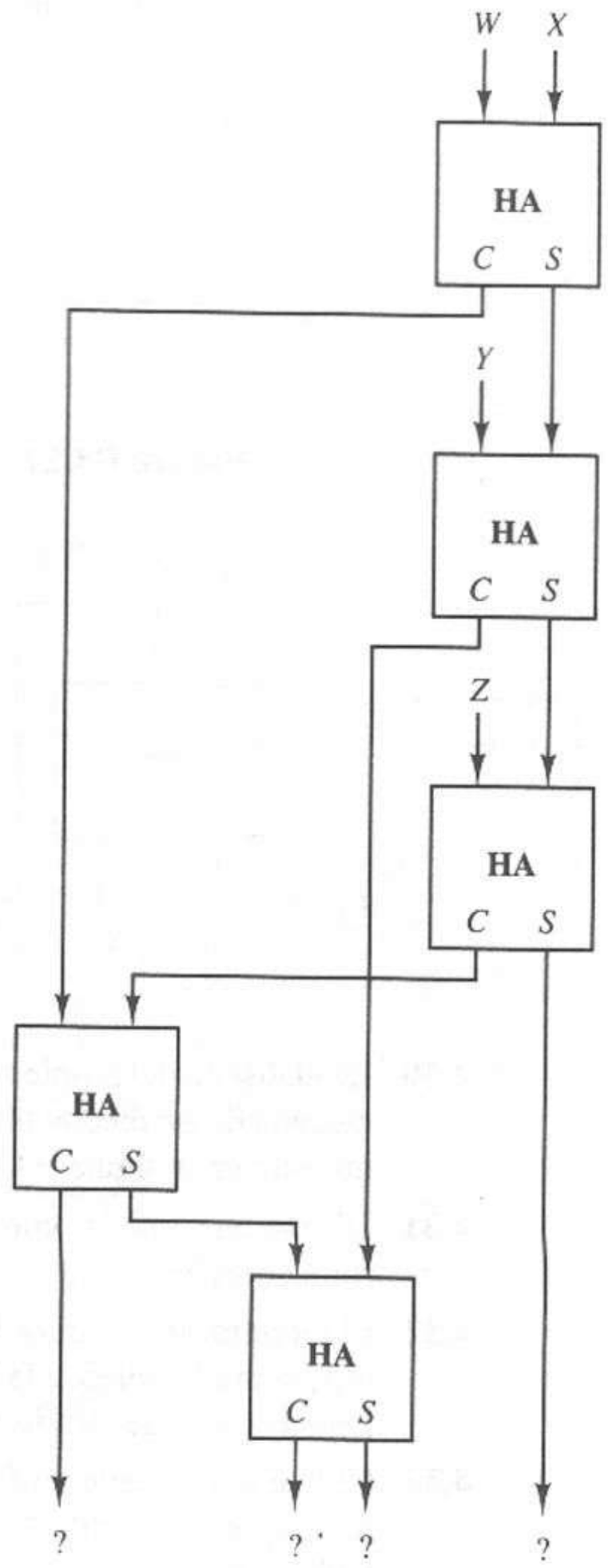


Figura P4.32

4.37 Diseñe un circuito de detección de desbordamiento para la figura 4.41, suponiendo que la unidad será utilizada para sumar y restar números en un sistema numérico de 4 bits complemento a dos.

4.38 El módulo sumador 7483 es más rápido que el 7482, debido a que el acarreo para cada etapa se calcula a partir de todas las entradas, en vez de propagarse a través de cada etapa.

(a) Determine la ecuación para el acarreo interno $C2$ del 7483 en términos únicamente de las entradas $A(i)$, $B(i)$ y $C0$.

(b) Si $A2A1$, $B2B1$, y $C0$ son los sumandos que determinan el acarreo (con $A2$ y $B2$ como MSB), determine las combinaciones de ($A2$, $A1$, $B2$, $B1$, $C0$) que producen $C_2 = 1$. (Determine los mintérminos para C_2.)

4.39 Diseñe un sumador BCD que sume dos dígitos BCD y produzca un resultado BCD y un acarreo de salida.

4.40 Diseñe un comparador de magnitud de 3 bits con entradas $A = (a_2a_1a_0)_2$ y $B = (b_2b_1b_0)_2$ y tres salidas: $EQ(A = B)$, $GT\ (A > B)$ y $LT\ (A < B)$.

4.41 Con las compuertas adecuadas y *un* comparador 7485, diseñe un circuito que compare dos números binarios de 5 bits $A = (a_4...a_0)$ y $B = (b_4...b_0)$, con $f_3 = 1$ cuando $A > B$, $f_2 = 1$ cuando $A = B$ y $f_1 = 1$ cuando $A < B$. (*Sugerencia*: Utilice las entradas en cascada y compuertas adicionales para comparar los dos dígitos menos significativos.)

4.42 Hay que comparar tres números de 4 bits $X = (x_3x_2x_1x_0)_2$, $Y = (y_3y_2y_1y_0)_2$ y $Z = (z_3z_2z_1z_0)_2$. Utilice comparadores de magnitud 7485 y las compuertas lógicas asociadas para diseñar un circuito que implante la siguiente tabla de verdad.

Condición	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
$X > Y > Z$	1	0	0	0	0	0	0	0
$X > Z > Y$	0	1	0	0	0	0	0	0
$Y > X > Z$	0	0	1	0	0	0	0	0
$Y > Z > X$	0	0	0	1	0	0	0	0
$Z > X > Y$	0	0	0	0	1	0	0	0
$Z > Y > X$	0	0	0	0	0	1	0	0
$X = Y = Z$	0	0	0	0	0	0	1	0
Cualquier otro caso	0	0	0	0	0	0	0	1

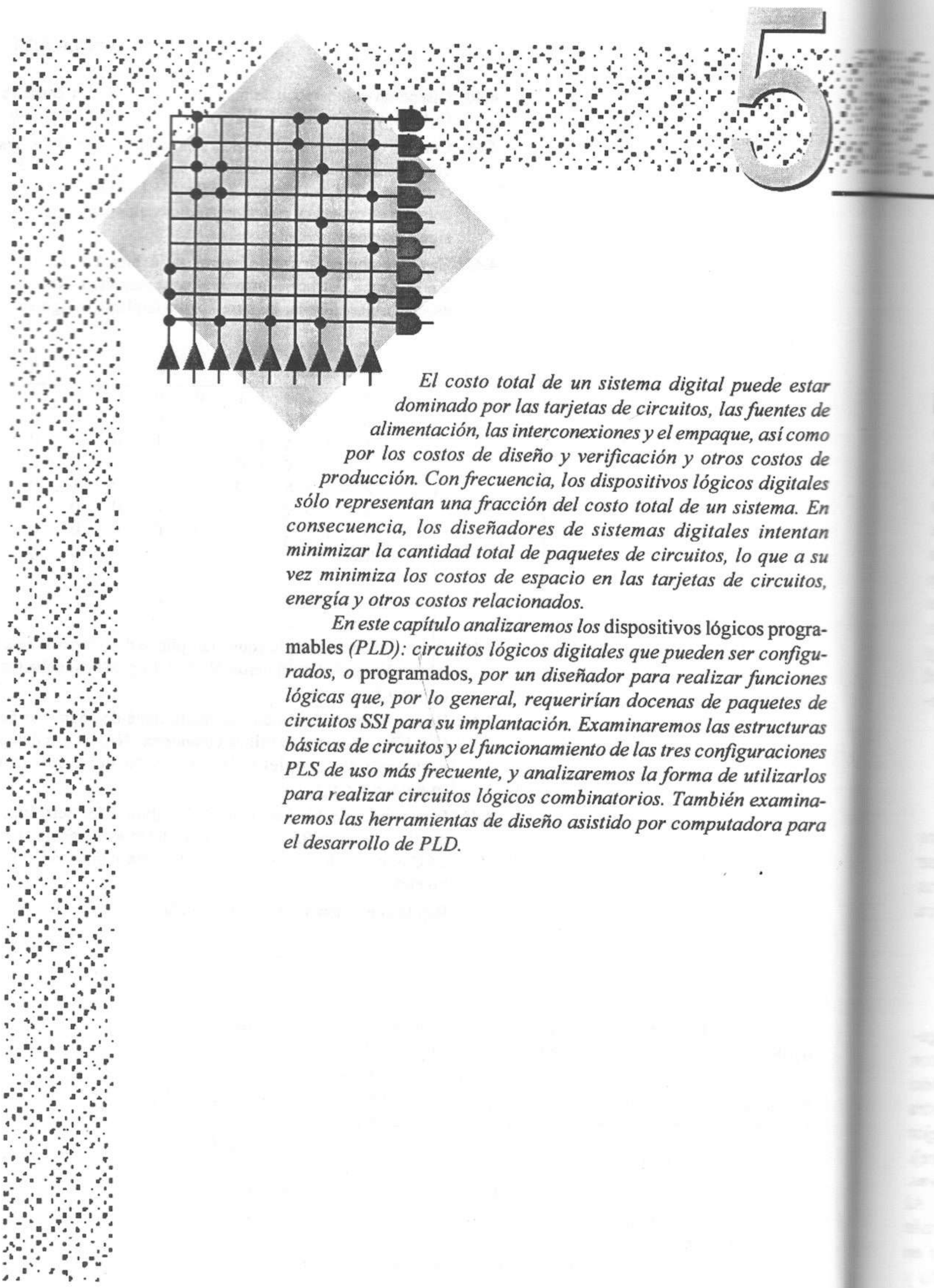
4.43 Diseñe un circuito lógico que multiplique dos números de 2 bits, $(a_1a_0)_2$ y $(b_1b_0)_2$, y que sólo utilice compuertas NAND. El producto debe ser un número de 4 bits $(p_3p_2p_1p_0)_2$.

4.44 Diseñe un circuito lógico que multiplique dos números de 4 bits, $(a_3a_2a_1a_0)_2$ y $(b_3b_2b_1b_0)_2$, y que sólo utilice compuertas NAND y módulos medios-sumadores y sumadores completos. El producto debe ser un número de 8 bits, $(p_7p_6p_5p_4p_3p_2p_1p_0)_2$.

4.45 Para la sección ALU de 1 bit de la figura 4.55, determine la forma de lista de mintérminos para $f_i(S_2, S_1, S_0, a_i, b_i, c_{i-1})$. Utilice el método tabular de minimización de Quine-McCluskey para determinar una implantación NAND mínima de dos niveles para f_i.

4.46 Repita el problema 4.45 para la salida C_i.

5



El costo total de un sistema digital puede estar dominado por las tarjetas de circuitos, las fuentes de alimentación, las interconexiones y el empaque, así como por los costos de diseño y verificación y otros costos de producción. Con frecuencia, los dispositivos lógicos digitales sólo representan una fracción del costo total de un sistema. En consecuencia, los diseñadores de sistemas digitales intentan minimizar la cantidad total de paquetes de circuitos, lo que a su vez minimiza los costos de espacio en las tarjetas de circuitos, energía y otros costos relacionados.

En este capítulo analizaremos los dispositivos lógicos programables *(PLD): circuitos lógicos digitales que pueden ser configurados, o* programados, *por un diseñador para realizar funciones lógicas que, por lo general, requerirían docenas de paquetes de circuitos SSI para su implantación. Examinaremos las estructuras básicas de circuitos y el funcionamiento de las tres configuraciones PLS de uso más frecuente, y analizaremos la forma de utilizarlos para realizar circuitos lógicos combinatorios. También examinaremos las herramientas de diseño asistido por computadora para el desarrollo de PLD.*

Diseño de un circuito combinatorio con dispositivos lógicos programables

5.1 Dispositivos lógicos semiadaptados

El número de paquetes de circuitos (circuitos integrados, IC) se reduce aumentando el nivel de integración, es decir, el número de compuertas por *chip*. Durante los últimos 20 años, el número de compuertas en un solo *chip* ha aumentado de unas cuantas compuertas en los dispositivos lógicos SSI estándar de la serie 7400 hasta más de un millón de dispositivos en los *chips* VLSI actuales de alto rendimiento. Un mayor nivel de integración implica por lo general un menor espacio en la tarjeta de circuitos impresos (PCB) y un menor consumo de energía.

Existen tres métodos básicos para implantar un circuito lógico digital LSI: con componentes estándar SSI, MSI y LSI, con dispositivos VLSI adaptados, y con dispositivos semiadaptados. Las funciones SSI, MSI y LSI estándar son convenientes, pues un circuito se puede ensamblar rápidamente con piezas disponibles en el mercado. Sin embargo, el total de componentes, y con ello el costo por compuerta, puede tener un tamaño inaceptable. La consolidación de un diseño en uno o más dispositivos adaptados o semiadaptados puede reducir el total de componentes, y con ello el costo total, de manera significativa.

En un diseño completamente adaptado, un IC se diseña compuerta por compuerta, desarrollando al mismo tiempo la disposición física de los componentes eléctricos y las conexiones entre ellos. Mediante las herramientas de diseño asistido por computadora (CAD), se puede optimizar el rendimiento del circuito y el aprovechamiento del área de silicio, aunque el proceso de diseño sea caro y largo. El uso de los dispositivos de circuito semiadaptados reduce el tiempo de diseño al utilizar arreglos de compuertas prediseñados, celdas estándar, o dispositivos lógicos programables. Un *arreglo de compuertas* es un IC que contiene varias compuertas lógicas no conectadas. El diseñador sólo tiene que especificar la forma de interconectar las compuertas del arreglo. La fabricación del dispositivo se divide en dos fases. En la fase inicial, las compuertas no conectadas se procesan y los *chips* se almacenan. La fase final sólo requiere fabricar las últimas capas de interconexión del IC, en vez de todo el circuito, lo que reduce el tiempo de fabricación de manera considerable. Se procesa un gran número de arreglos hasta llegar a las últimas capas de interconexión y se guardan entonces en espera de una personalización para cierta aplicación.

Una *celda estándar* es un circuito desarrollado en una *biblioteca de diseño* junto con otras celdas estándar. El diseñador crea un IC eligiendo celdas de la biblioteca, especificando su posición en el IC, e indicando después la forma de

interconectarlas. Este proceso no es muy distinto del diseño de una tarjeta de circuitos impresos con paquetes SSI/MSI/LSI. La complejidad de las funciones de una biblioteca de celdas estándar puede variar desde compuertas lógicas discretas hasta microprocesadores completos y otros circuitos complejos. Un diseño con celdas estándar es con frecuencia menos óptimo que uno completamente adaptado, pero el tiempo de diseño se reduce de manera considerable. El IC resultante se fabrica desde cero, como si fuese un diseño completamente adaptado; es decir, se necesitan todos los pasos de procesamiento para personalizar el circuito para cierta aplicación.

Los pasos de fabricación para personalizar un dispositivo pueden evitarse usando *dispositivos lógicos programables* (PLD). Los PLD son IC prefabricados que también incluyen capas de interconexión flexibles. Las capas de interconexión se personalizan por medios electrónicos para una aplicación específica. En muchos casos, el usuario final puede realizar esta personalización electrónica. Los PLD contienen los recursos necesarios para realizar las expresiones de conmutación básicas de dos niveles e incluyen con frecuencia otros elementos lógicos, con el equivalente de varios miles de compuertas lógicas en un solo PLD. Un circuito PLD se desarrolla mediante el diseñando de expresiones lógicas, traduciéndolas al formato del PLD objetivo, e instalándolas en el PLD mediante un programador de PLD. Así, se puede producir un dispositivo funcional en unos cuantos minutos, en vez de los días o semanas necesarios para fabricar un IC con un arreglo de compuertas o celdas estándar. En caso necesario, se pueden implantar cambios de diseño en forma rápida y barata en horas o incluso minutos, mientras que los cambios de diseño en el caso de las celdas estándar o los arreglos de compuertas requieren una refabricación completa y, por ello, tardan días o semanas.

5.2 Circuitos de arreglos lógicos

Los circuitos de arreglos lógicos programables se construyen con base en arreglos homogéneos de componentes elementales que pueden configurarse para realizar operaciones lógicas AND y OR. En esta sección analizaremos las estructuras básicas y el funcionamiento de estos circuitos, incluyendo los mecanismos para que un diseñador pueda programarlos y realizar funciones de conmutación.

5.2.1 Operación de diodos en circuitos digitales

Los dispositivos lógicos programables se construyen con diversas configuraciones de diodos semiconductores, interruptores con transistores o elementos similares. Un *diodo PN* es un dispositivo electrónico formado mediante la unión de dos tipos de materiales semiconductores, el tipo *p* y el tipo *n*, como se muestra en la figura 5.1. Un *semiconductor* es un material que conduce la corriente mejor que un aislante (como el hule), pero menos que un conductor (como el cobre). Los tipos *p* y *n* indican mecanismos de conducción de cargas positivas y negativas. El funcionamiento de un diodo puede aproximar el de un interruptor "ideal". Al aplicar un voltaje a través del diodo y hacer que el lado *p* (el ánodo) sea más positivo que el lado *n* (el cátodo), se dice que el diodo tiene *polarización en directa*. En este modo, se comporta como un interruptor o circuito cerrado y mantiene esta polarización en un valor muy pequeño (es decir, el voltaje es casi nulo). Cuando el cátodo es mucho más positivo que el ánodo, se dice que el

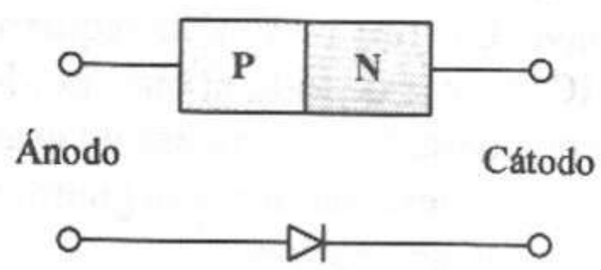


Figura 5.1 Diodo de unión PN y su símbolo esquemático.

diodo tiene *polarización en inversa*, y se comporta de hecho como un circuito abierto (de modo que el voltaje total aparece a través del diodo).

En las aplicaciones de circuitos digitales, se conecta una terminal de cada diodo a la fuente de alimentación o bien a tierra mediante una resistencia, como en las figuras 5.2a y d. La otra terminal se controla mediante una señal lógica digital que establece la polarización del diodo. Consideremos el funcionamiento del circuito de la figura 5.2a. Cuando la señal A es 1 lógico (un voltaje positivo), el diodo tiene polarización en inversa y se comporta como un circuito abierto, como se muestra en la figura 5.2b. En este caso, la línea de señal B es llevada por la resistencia hacia el voltaje V de la fuente de alimentación, lo que hace que la señal B sea igual a 1 lógico. Cuando A es 0 lógico (0 volts), el diodo tiene polarización en directa y, por tanto, comienza a conducir, comportándose como un circuito cerrado, como se muestra en la figura 5.2c, y haciendo que el voltaje en B sea 0 lógico. Las figuras 5.2d, e y f ilustran los efectos equivalentes cuando la resistencia se conecta a tierra.

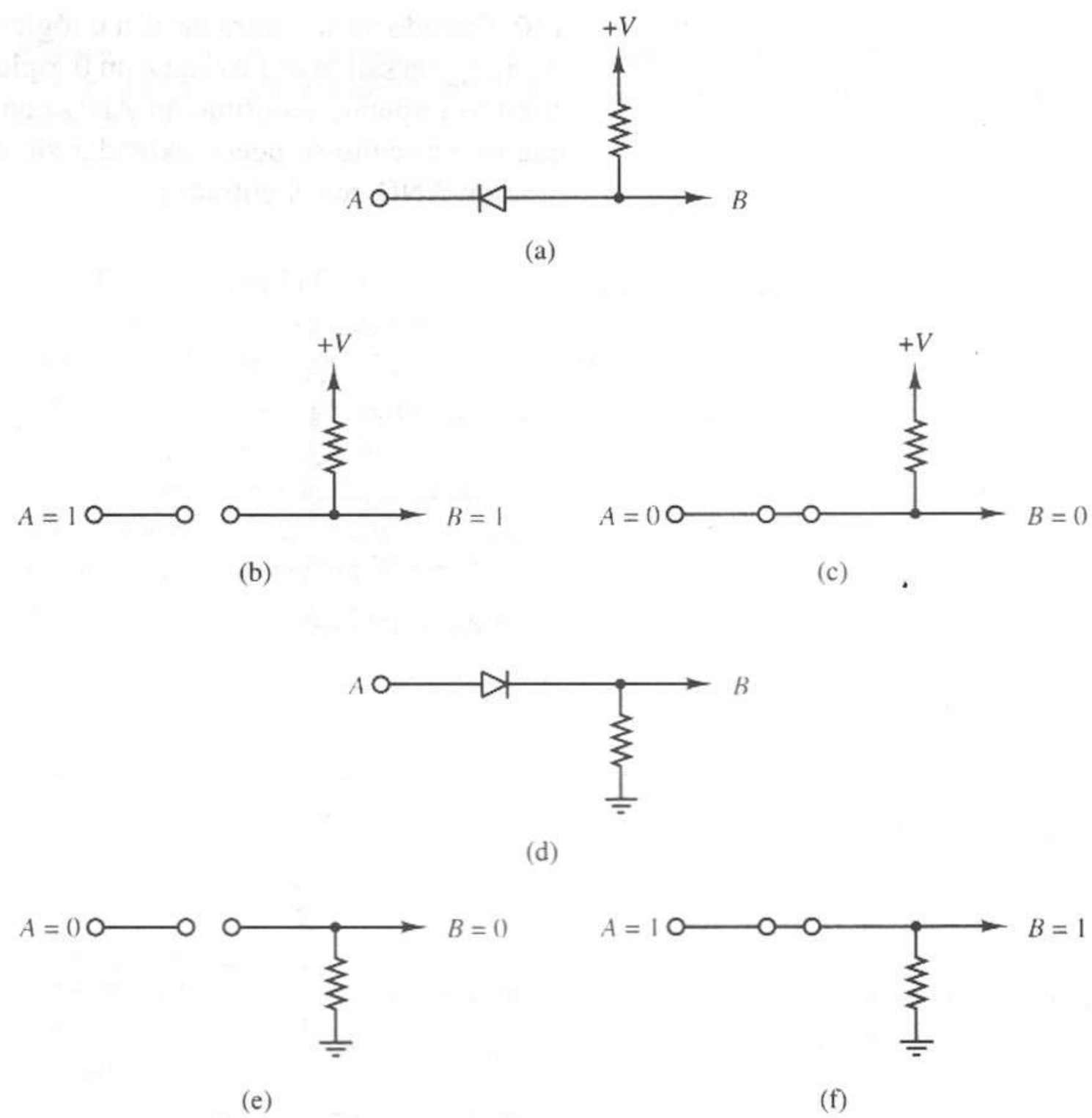


Figura 5.2 Funcionamiento del diodo PN en aplicaciones digitales. **(a)** Con resistencia *pull-up* (elevadora). **(b)** Con polarización en inversa: diodo abierto; B debe asumir el valor 1. **(c)** Con polarización en directa: diodo cerrado, B debe asumir el valor 0. **(d)** Con resistencia *pull-down* (reductora). **(e)** Con polarización en inversa: diodo abierto; B debe asumir el valor 0. **(f)** Con polarización en directa: diodo cerrado, B debe asumir el valor 1.

En las aplicaciones lógicas programables se utilizan muchos otros dispositivos electrónicos, como los interruptores de transistores, en vez de diodos. Aunque en este capítulo analizaremos de manera específica los circuitos lógicos construidos con diodos, los conceptos de diseño lógico digital son los mismos para los otros dispositivos. El lector puede consultar textos de electrónica [1] o de dispositivos lógicos programables [2,3] si desea mayor información acerca de estos dispositivos.

5.2.2 Arreglos lógicos AND y OR

El comportamiento arriba descrito permite implantar fácilmente las funciones de conmutación con diodos PN. Por ejemplo, el circuito de la figura 5.3a realiza una función AND con tres entradas, A, B y C. Verifiquemos esto obteniendo la tabla de verdad de este circuito. Cuando $ABC = 111$, los tres diodos están abiertos, y la salida es llevada a un 1 lógico, como se muestra en la figura 5.3b. Si A cambia a 0, el diodo correspondiente se cierra, lo cual hace que el voltaje de salida sea 0, como en la figura 5.3c. Como los otros dos diodos siguen abiertos, no afectan la salida. Por simetría, lo mismo es cierto para $ABC = 101$ y $ABC = 110$. Cuando varias entradas son 0 lógico, todos los diodos correspondientes se cierran, y la salida es forzada a un 0 lógico, como en la figura 5.3d. Por tanto, el circuito implanta una función AND con tres entradas. El lector debe verificar que este circuito se puede extender sin dificultad a K diodos para realizar una función AND con K entradas.

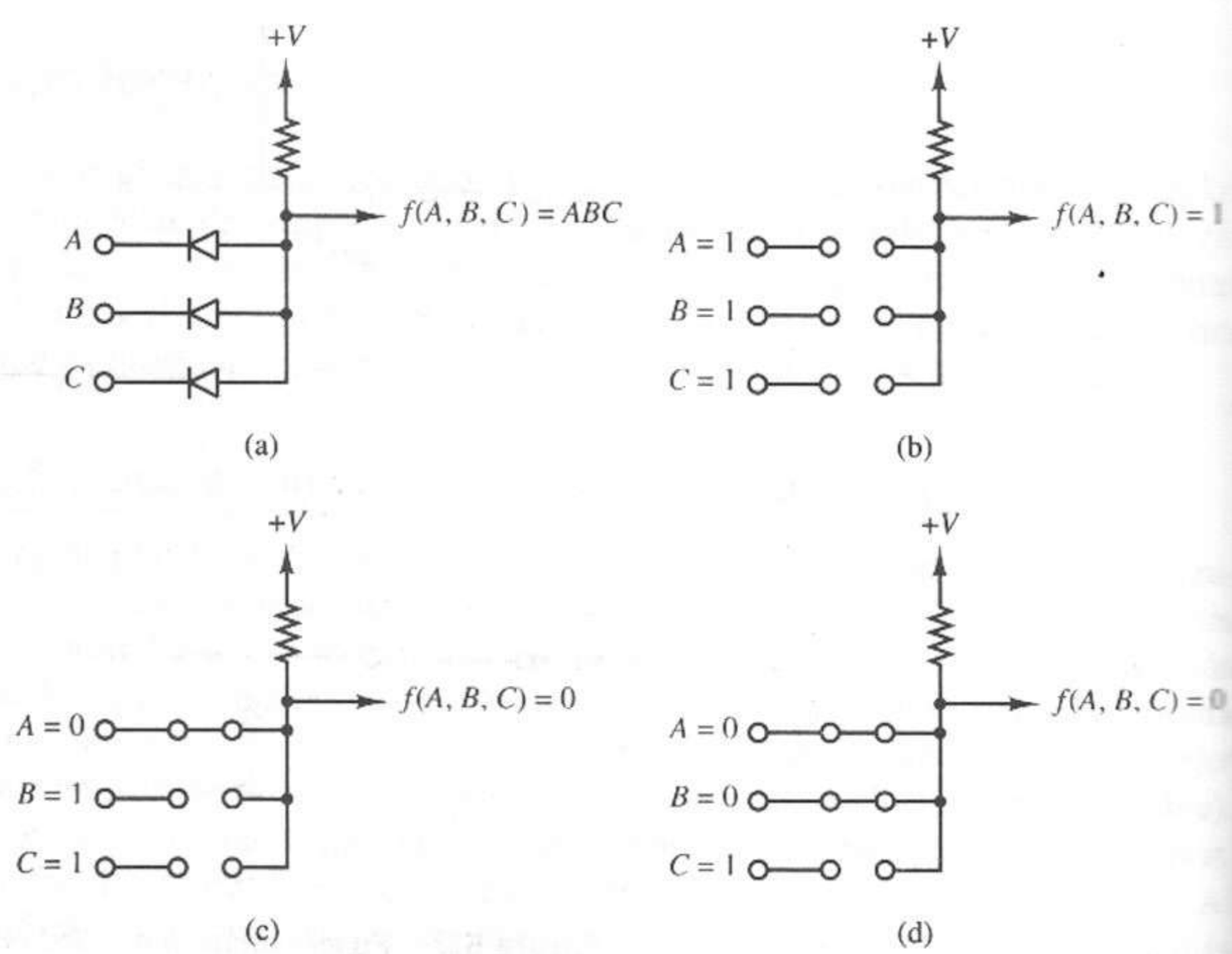


Figura 5.3 Función AND realizada mediante un arreglo de diodos. **(a)** Configuración básica. **(b)** Todos los diodos abiertos; f debe asumir el valor 1. **(c)** Un diodo cerrado; f debe asumir el valor 0. **(d)** Varios diodos cerrados, f debe asumir el valor 0.

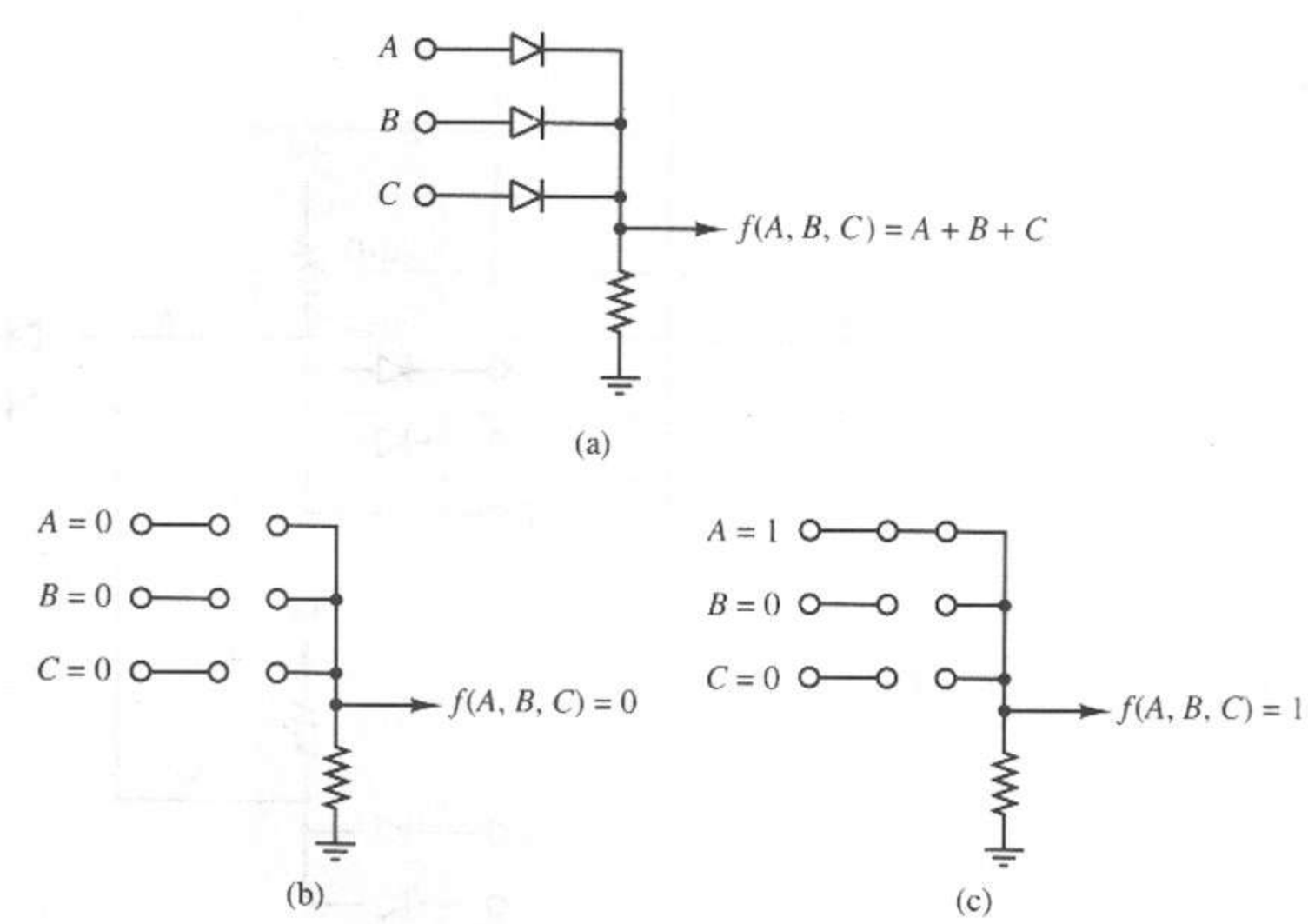


Figura 5.4 Función OR realizada con un arreglo de diodos. **(a)** Configuración básica. **(b)** Todos los diodos abiertos; f debe asumir el valor 0. **(c)** Un diodo cerrado, f debe asumir el valor 1.

El circuito de la figura 5.4a realiza una función OR. En este circuito, cuando $ABC = 000$, los tres diodos están abiertos y la salida es llevada a un 0 lógico, como en la figura 5.4b. Cuando $ABC = 100$, como en la figura 5.4c, el diodo conectado a A conduce (se comporta como un circuito cerrado), lo que lleva la salida a un 1 lógico. Como los otros dos diodos siguen abiertos, no afectan la salida. De manera análoga, la salida será 1 lógico cuando cualquier otra entrada, o combinación de entradas, sea igual a 1 lógico. Así, el circuito realiza una función OR con tres entradas. De nuevo, observe que este circuito se puede extender fácilmente a una función de K entradas mediante K diodos.

5.2.3 Arreglos AND-OR de dos niveles

Podemos conectar entre sí los circuitos AND y OR ya descritos de la misma manera que las compuertas lógicas para realizar cualquier función de conmutación. Por ejemplo, consideremos la función

$$f(a, b, c) = ab\bar{c} + \bar{b}c$$

que está en forma de suma de productos de dos niveles. La figura 5.5a muestra una realización de esta función con arreglos lógicos de diodos. La figura 5.5b muestra un formato más compacto, que se utiliza por lo general para producir circuitos con arreglos lógicos de diodos. Las funciones AND se representan mediante líneas verticales con símbolos de compuertas AND que representan las resistencias de *pull-up* (elevadoras), y la función OR se representa mediante una línea horizontal con un símbolo de compuerta OR, lo que representa la resistencia de *pull-down* (reductora). El símbolo × representa los diodos. A veces omitimos los símbolos AND para abreviar.

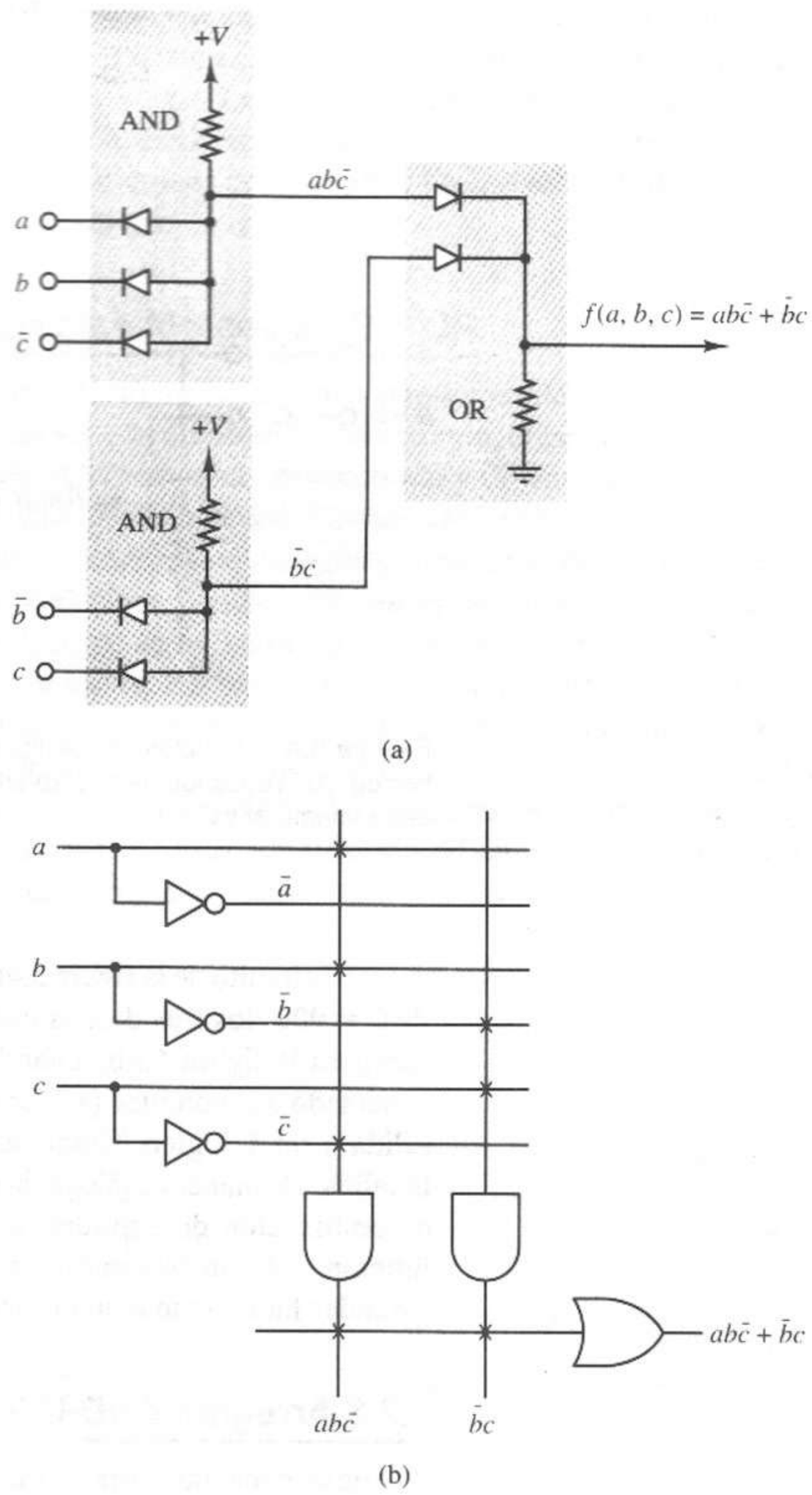


Figura 5.5 Formas suma de productos realizadas con arreglos AND-OR. **(a)** Conexión de los arreglos AND y OR. **(b)** Forma compacta.

La figura 5.6 muestra la forma de realizar varias funciones con un único arreglo lógico, agregando circuitos OR. En este ejemplo, realizamos las siguientes dos funciones:

$$f_1(a, b, c) = ab + \bar{c}$$
$$f_2(a, b, c) = ab + \bar{b}c$$

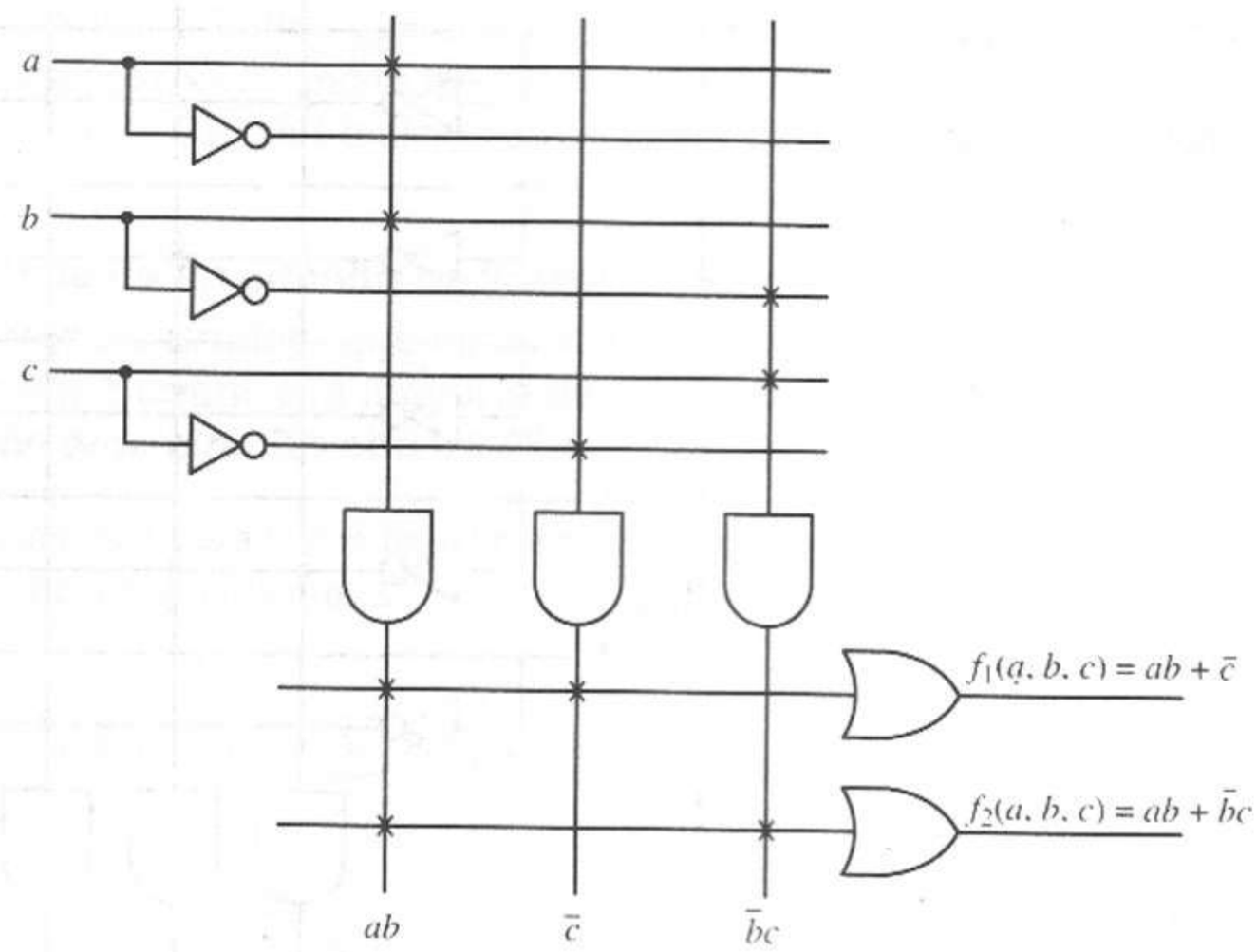


Figura 5.6 Varias funciones realizadas en un arreglo AND-OR.

Observe que el término producto *ab* se utiliza en ambas funciones, es decir, es "compartido" por ambas operaciones OR. La especificación de las posiciones de los diodos en el arreglo personaliza, o programa, el arreglo para una función lógica específica.

La combinación de un arreglo AND programable seguido de un arreglo OR programable se conoce como *arreglo lógico programable (PLA)*, ya que se pueden realizar funciones lógicas especificando (programando) la configuración de los diodos.

EJEMPLO 5.1

Diseñar un PLA que realice las siguientes tres funciones lógicas y muestre las conexiones internas.

$$f_1(A,B,C,D,E) = \bar{A}\bar{B}\bar{D} + \bar{B}C\bar{D} + \bar{A}BCD\bar{E}$$
$$f_2(A,B,C,D,E) = \bar{A}BE + \bar{B}C\bar{D}E$$
$$f_3(A,B,C,D,E) = \bar{A}\bar{B}\bar{D} + \bar{B}\bar{C}\bar{D}E + \bar{A}BCD$$

Como hay cinco variables, el PLA debe tener cinco entradas, cada una de las cuales debe aparecer complementada y no complementada. Hay un total de siete términos producto únicos en las tres expresiones anteriores. Por tanto, el PLA debe generar al menos siete términos producto. Por último, como queremos realizar tres funciones, hay que generar tres términos suma (OR).

La figura 5.7 muestra la organización del PLA. La tabla 5.1 muestra las conexiones por realizar en los arreglos AND y OR. En la tabla, los números de los términos producto corresponden a los números de compuerta AND en la figura 5.7, cada uno de los cuales se conecta a una *línea producto* vertical, sobre la que se genera un término producto. En la parte de la tabla correspondiente al

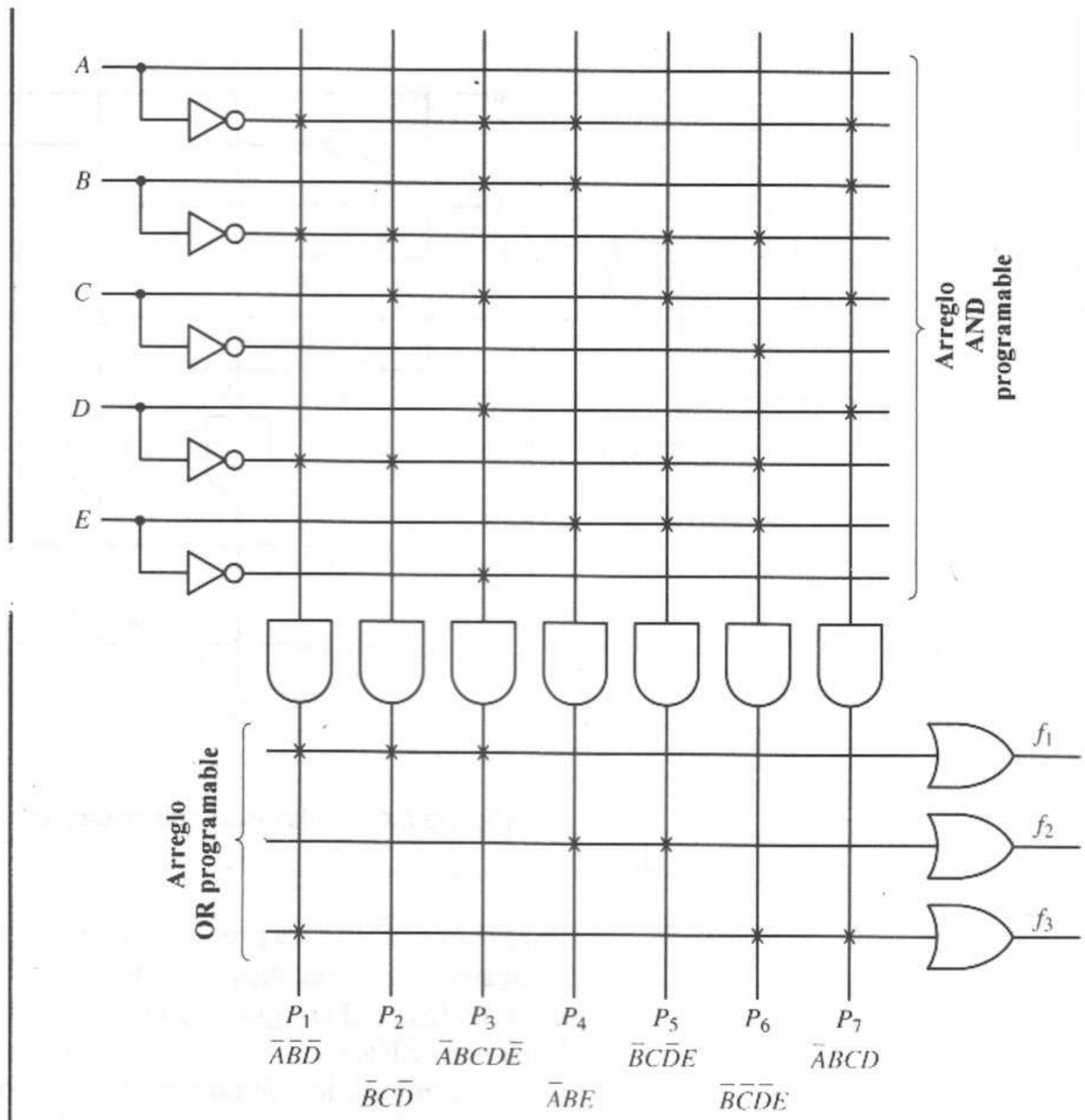


Figura 5.7 PLA para el ejemplo 5.1.

Tabla 5.1 TABLA PLA PARA EL EJEMPLO 5.1

Término producto	Entradas del arreglo AND $ABCDE$	Salidas del arreglo OR $f_1f_2f_3$
1 $\bar{A}\bar{B}\bar{D}$	00×0×	1 0 1
2 $\bar{B}C\bar{D}$	×010×	1 0 0
3 $\bar{A}BCD\bar{E}$	01110	1 0 0
4 $\bar{A}BE$	01××1	0 1 0
5 $\bar{B}C\bar{D}E$	×0101	0 1 0
6 $\bar{B}\bar{C}\bar{D}E$	×0001	0 0 1
7 $\bar{A}BCD$	0111×	0 0 1

arreglo AND, un 0 indica que se conecta el complemento de la variable a la línea producto, un 1 indica que la entrada no complementada se conecta a la línea producto, y un × indica que ninguna de ellas se conecta a la línea producto. Para el arreglo OR, un 1 indica una conexión y un 0 indica que no hay conexión.

EJEMPLO 5.2

La salida de un circuito de votación por mayoría con cinco entradas, que aparece en la figura 5.8a, debe ser 1 cuando la mayoría de sus entradas es 1. Diseñe este circuito con un PLA.

La salida es 1 para todos los mintérminos que contienen tres o más unos. La función correspondiente es

$$f(a, b, c, d, e) = \sum m(7, 11, 13, 14, 15, 19, 21, 22, 23, 25 - 31)$$

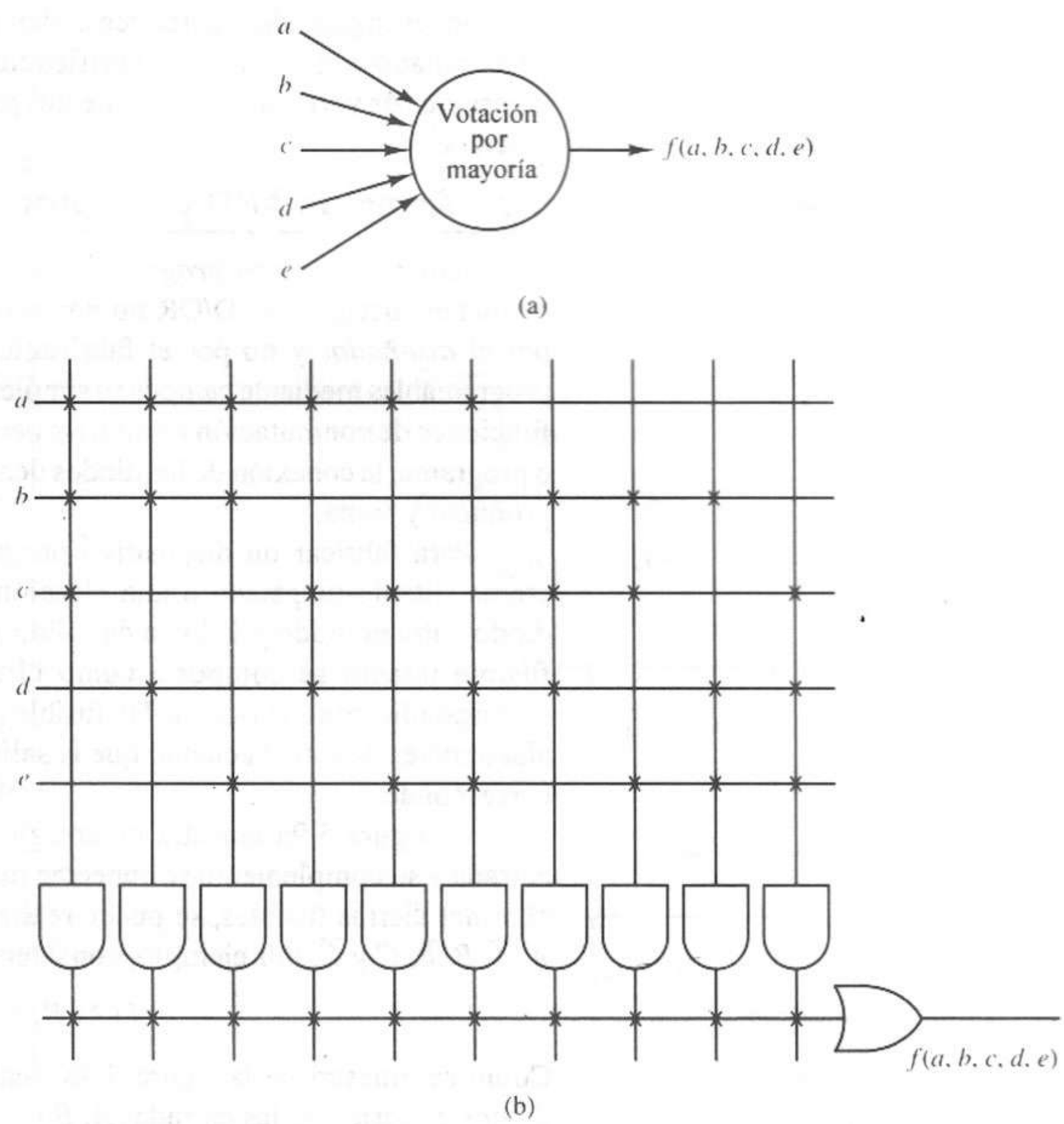


Figura 5.8 Votación por mayoría del ejemplo 5.2. **(a)** Salida = 1 si la mayoría de entradas = 1. **(b)** Realización PLA.

Mediante un mapa K o algún otro método adecuado, podemos expresar la función en forma de suma de productos mínima como

$$f(a, b, c, d, e) = abc + abd + abe + acd + ace + ade + bcd + bce + bde + cde$$

El PLA de la figura 5.8b realiza esta función.

Los diseñadores de circuitos VLSI adaptados o con celdas estándar utilizan con frecuencia arreglos lógicos programables en vez de compuertas lógicas discretas para las partes de lógica combinatoria de sus diseños. Las conexiones en los arreglos AND y OR se crean durante el proceso de fabricación del dispositivo (es decir, se programan con una plantilla). La configuración de cada PLA, es decir, el número de entradas, términos producto y términos suma, se adapta al circuito específico por realizar. El uso de los PLA tiene varias ventajas. En primer lugar, los PLA pueden hacerse más compactos que los circuitos equivalentes de compuertas lógicas discretas, lo que reduce el área del circuito. En segundo lugar, existen herramientas de diseño asistido por computadora que pueden generar en forma automática diseños físicos de PLA directamente de las ecuaciones lógicas, lo que reduce el tiempo global de diseño. Por último, se han desarrollado algoritmos de verificación para PLA más eficientes que los algoritmos de verificación equivalentes para los circuitos con compuertas lógicas discretas.

5.2.4 Arreglos AND y OR programables mediante campos

Los elementos lógicos *programables mediante campos* son dispositivos que contienen arreglos AND/OR no comprometidos, programados (configurados) por el *diseñador* y no por el fabricante. Muchos arreglos AND/OR estándar programables mediante campos (o simplemente "programables") pueden realizar funciones de conmutación arbitrarias permitiendo al diseñador que especifique, o programe, la conexión de los diodos dentro de los arreglos para formar términos producto y suma.

Para fabricar un dispositivo programable, un fusible metálico (níquel-cromo, titanio-tungsteno o una aleación similar) se coloca en serie con cada diodo entre el diodo y la línea de salida, como se muestra en la figura 5.9a. Un fusible intacto se comporta como circuito cerrado, que conecta el diodo correspondiente con la salida. Un fusible se puede eliminar pasando una corriente alta a través de éste, haciendo que la salida sea independiente de la entrada que corresponde.

La figura 5.9a muestra un arreglo programable AND. Cada variable de entrada y su complemento se conectan mediante diodos y fusibles a la salida. Al eliminar ciertos fusibles, se puede realizar cualquier producto de las variables A, $\overline{A}$, B, $\overline{B}$, C y $\overline{C}$. Por ejemplo, consideremos la función de conmutación

$$f(A, B, C) = \bar{A}B\bar{C}$$

Como se muestra en la figura 5.9b, realizamos este producto eliminando los diodos en serie con las entradas A, $\overline{B}$ y C y dejando intactos los diodos en serie con las entradas $\overline{A}$, B y $\overline{C}$. De manera análoga, la función

$$f(A, B, C) = A\bar{B}$$

se realiza eliminando cuatro fusibles, como en la figura 5.9c.

En vez de dibujar cada diodo y fusible de forma individual, utilizamos por lo general una notación abreviada para representar configuraciones lógicas

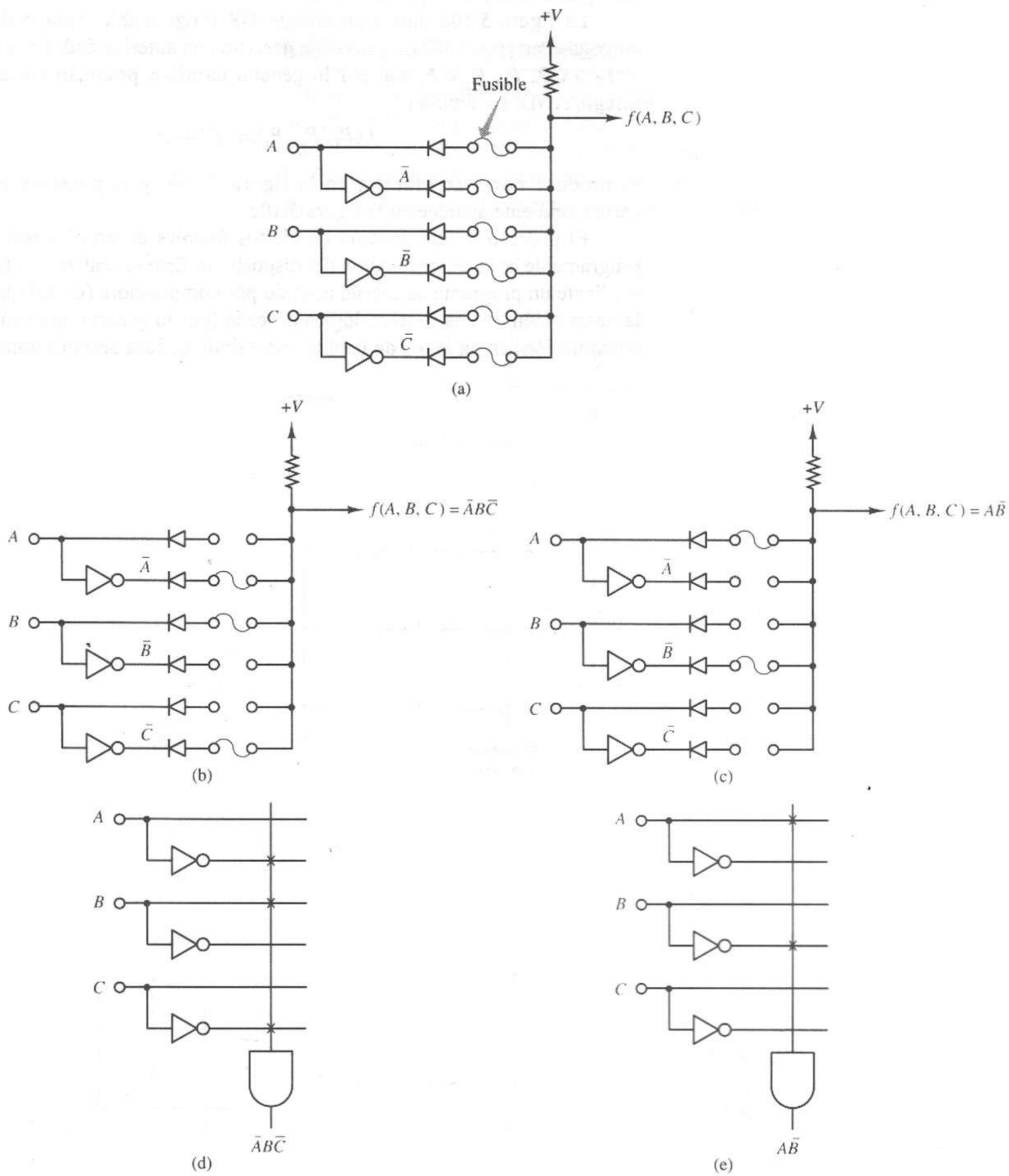


Figura 5.9 Arreglos AND programables mediante fusibles. **(a)** Arreglo AND no programado. **(b)** $f(A, B, C) = \bar{A}B\bar{C}$. **(c)** $f(A, B, C) = A\bar{B}$. **(d)** Notación compacta para (b). **(e)** Notación compacta para (c).

programables, como en las figuras 5.9d y e. Una × colocada en una intersección representa un fusible, y la ausencia de la × indica la ausencia de un fusible. El lector debe verificar que los circuitos de las figuras 5.9d y e representan los de las figuras 5.9b y c, respectivamente.

La figura 5.10a ilustra un arreglo OR programable, creado de manera análoga al arreglo AND programable descrito con anterioridad. Las entradas al arreglo OR, P_1, P_2 y P_3 son por lo general términos producto creados en un arreglo AND. La función

$$f(P_1, P_2, P_3) = P_1 + P_3$$

se produce como se muestra en la figura 5.10b, y la notación abreviada correspondiente aparece en la figura 5.10c.

El proceso de eliminación de ciertos fusibles de un dispositivo lógico programable es la *programación* del dispositivo. Esto se realiza con frecuencia mediante un programa de diseño asistido por computadora (CAD) que traduce la descripción de una función lógica deseada (por lo general una expresión de conmutación) en un *mapa* de fusibles por eliminar, para después transferir este

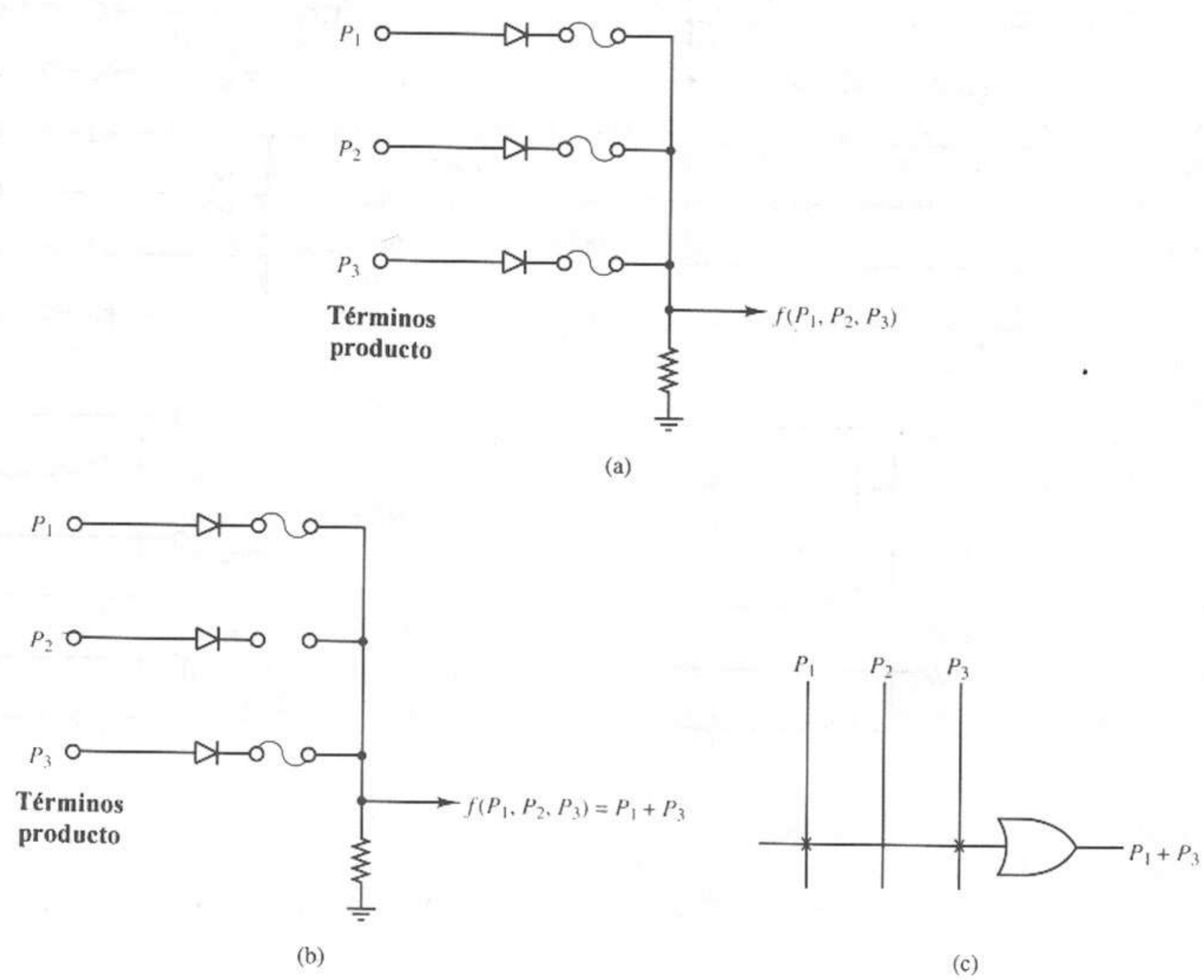


Figura 5.10 Arreglos OR programables mediante fusibles. **(a)** Arreglo OR no programado. **(b)** $f(P_1, P_2, P_3) = P_1 + P_3$. **(c)** Forma compacta.

mapa de fusibles a un instrumento especial llamado *programador de dispositivos*, el cual selecciona los fusibles especificados en el mapa y después proporciona la corriente para eliminarlos.

5.2.5 Opciones de polaridad de salida

Además de los términos producto y suma, los dispositivos lógicos programables tienen con frecuencia algunas otras características estándar, como polaridad de salida programable, señales de retroalimentación y *pines* de señal bidireccional. La figura 5.11a muestra las opciones de polaridad de salida comunes disponibles en los PLD: polaridad alta activa, baja activa, complementaria y programable.

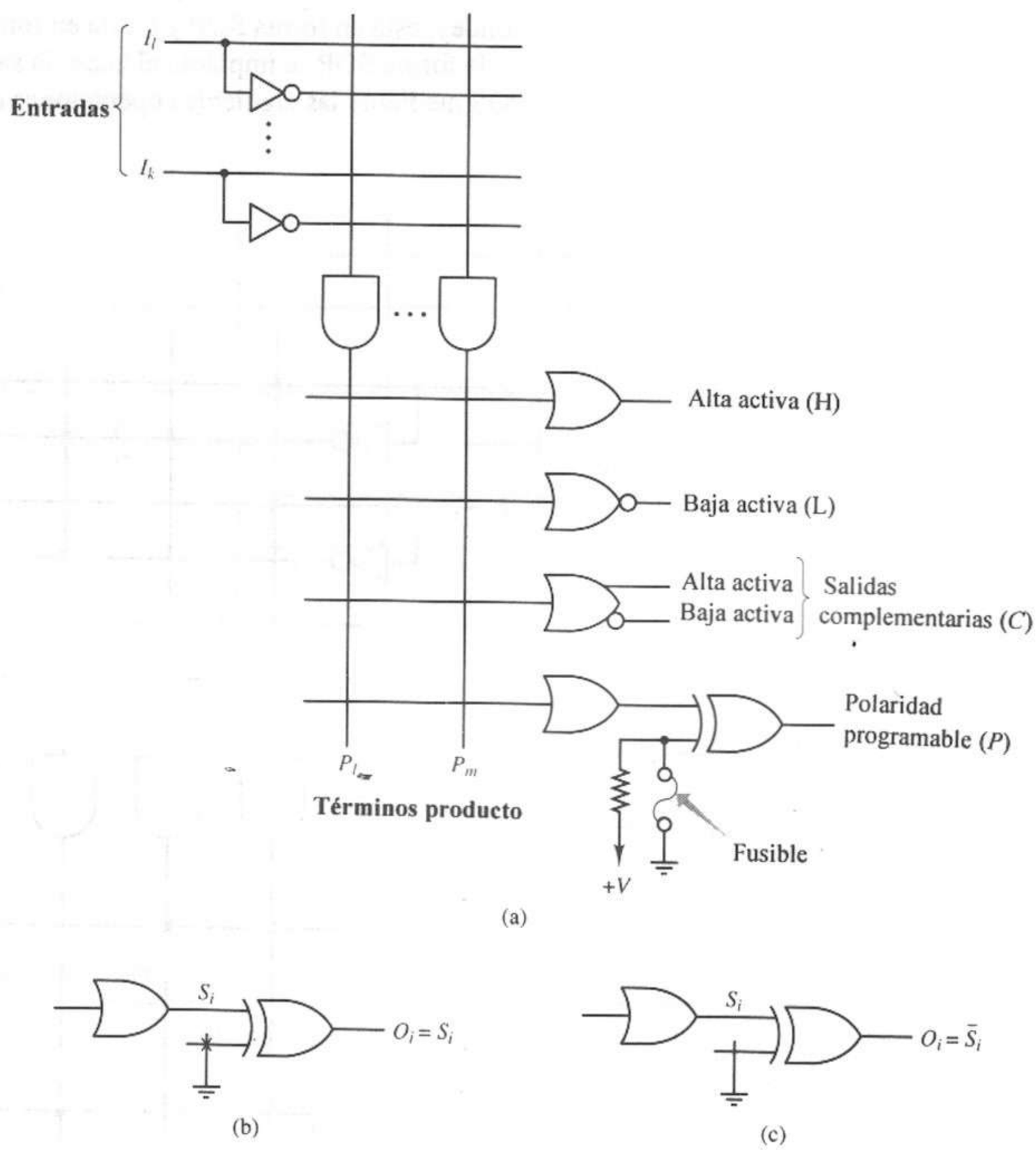


Figura 5.11 Opciones de polaridad de salida para los dispositivos lógicos programables mediante campos. **(a)** Opciones de polaridad de salida. **(b)** Fusible intacto: O_i es alta activa, $O_i = S_i \otimes 0 = S_i$. **(c)** Fusible eliminado: O_i es baja activa, $O_i = S_i \otimes 1 = \bar{S}_i$.

Creamos una salida de polaridad programable mediante una compuerta OR exclusiva (XOR) con una entrada de fusible. Esta entrada tiene el valor 0 lógico cuando el fusible se deja intacto o 1 lógico cuando el fusible es eliminado. Si recordamos el funcionamiento de la compuerta XOR, cuando el fusible queda intacto, como en la figura 5.11b, la salida $Oi = Si \oplus 0 = S_i$ y entonces la salida es alta activa. Cuando el fusible se elimina, como en la figura 5.11c, la salida $Oi = S_i \oplus 1 = S_i$, y entonces la salida es baja activa. Observe que una × indica un fusible intacto y la ausencia de una × indica un fusible eliminado, como en la convención para los arreglos AND/OR.

Además de permitir la producción de salidas altas activas y bajas activas, la polaridad de salida programable permite realizar las formas suma de productos (SOP) y producto de sumas (POS). Por ejemplo, la figura 5.12 ilustra la realización de las siguientes funciones:

$$f_1(A, B, C) = AB + \bar{A}C$$

$$f_2(A, B, C) = (A + B)(\bar{A} + C)$$

donde f_1 está en forma SOP y f_2 está en forma POS. Como se muestra en la figura, la forma SOP se implanta al hacer la salida activa alta. Realizamos la forma POS mediante las siguientes operaciones algebraicas.

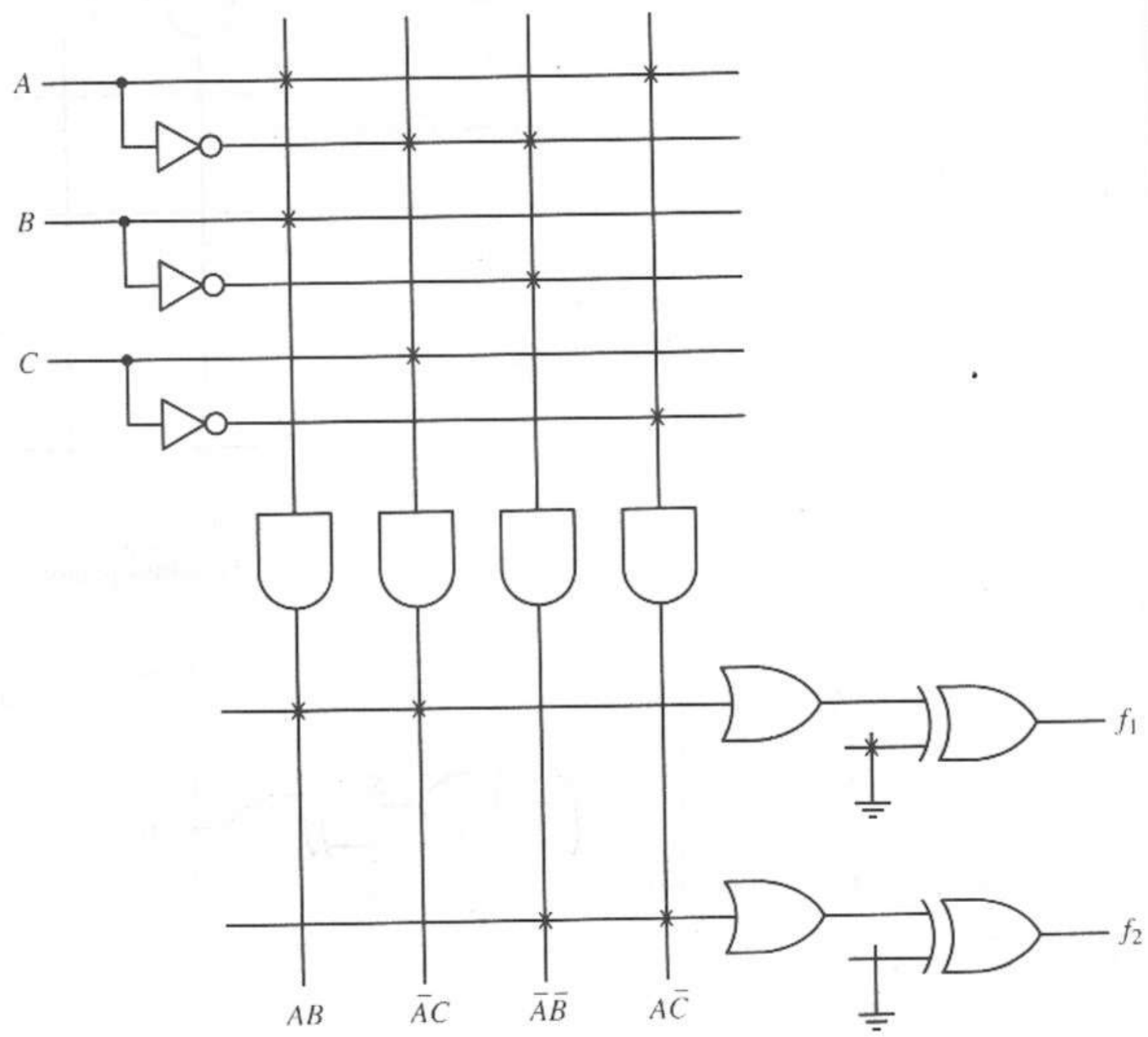


Figura 5.12 Implantación de formas SOP y POS mediante polaridad de salida programable.

$$
\begin{aligned}
(A+B)(\bar{A}+C) &= \overline{\overline{(A+B)(\bar{A}+C)}} \\
&= \overline{\overline{(A+B)} + \overline{(\bar{A}+C)}} \\
&= \overline{\bar{A}\bar{B} + A\bar{C}}
\end{aligned}
$$

Así, podemos realizar la forma POS de manera indirecta invirtiendo una expresión SOP. Por tanto, los términos producto y la polaridad de salida programable permiten realizar cualesquiera de las compuertas lógicas primitivas (AND, OR, NAND, NOR o NOT).

5.2.6 Pines bidireccionales y líneas de retroalimentación

Otra característica de muchos dispositivos lógicos programables mediante campos es un *pin* de entrada/salida bidireccional, como el que se ilustra en la

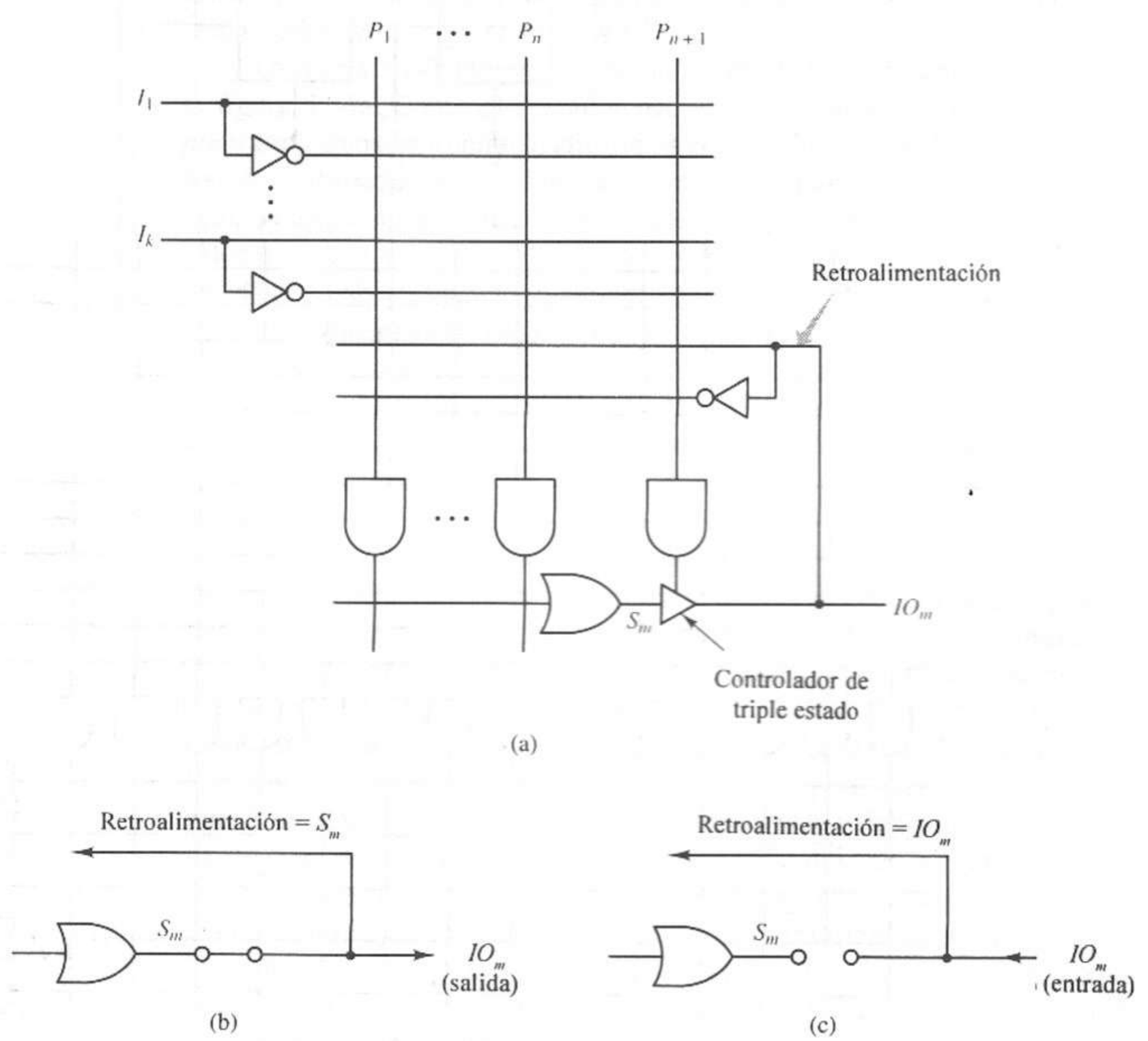


Figura 5.13 *Pines* bidireccionales en dispositivos lógicos programables. **(a)** IO_m es un *pin* de entrada/salida bidireccional. **(b)** Controlador de triple estado activado ($P_{n+1} = 1$). **(c)** Controlador de triple estado desactivado ($P_{n+1} = 0$).

figura 5.13a. Controlamos un *pin* bidireccional mediante un *controlador de tres estados*, cuya línea de control está conectada a uno de los términos producto. Cuando la línea de control tiene el valor 1, se dice que el controlador está *activo* y funciona como circuito cerrado (o interruptor cerrado), como en la figura 5.13b. En este caso, el término suma se guía hacia el *pin*, que funciona como una salida. Además, este valor es retroalimentado al arreglo AND, donde se puede utilizar para formar términos producto. De esta manera, podemos realizar circuitos de varios niveles (más que 2).

Cuando la línea de control del controlador tiene el valor 0, el controlador está *inactivo* y funciona como un circuito abierto (o interruptor abierto), como

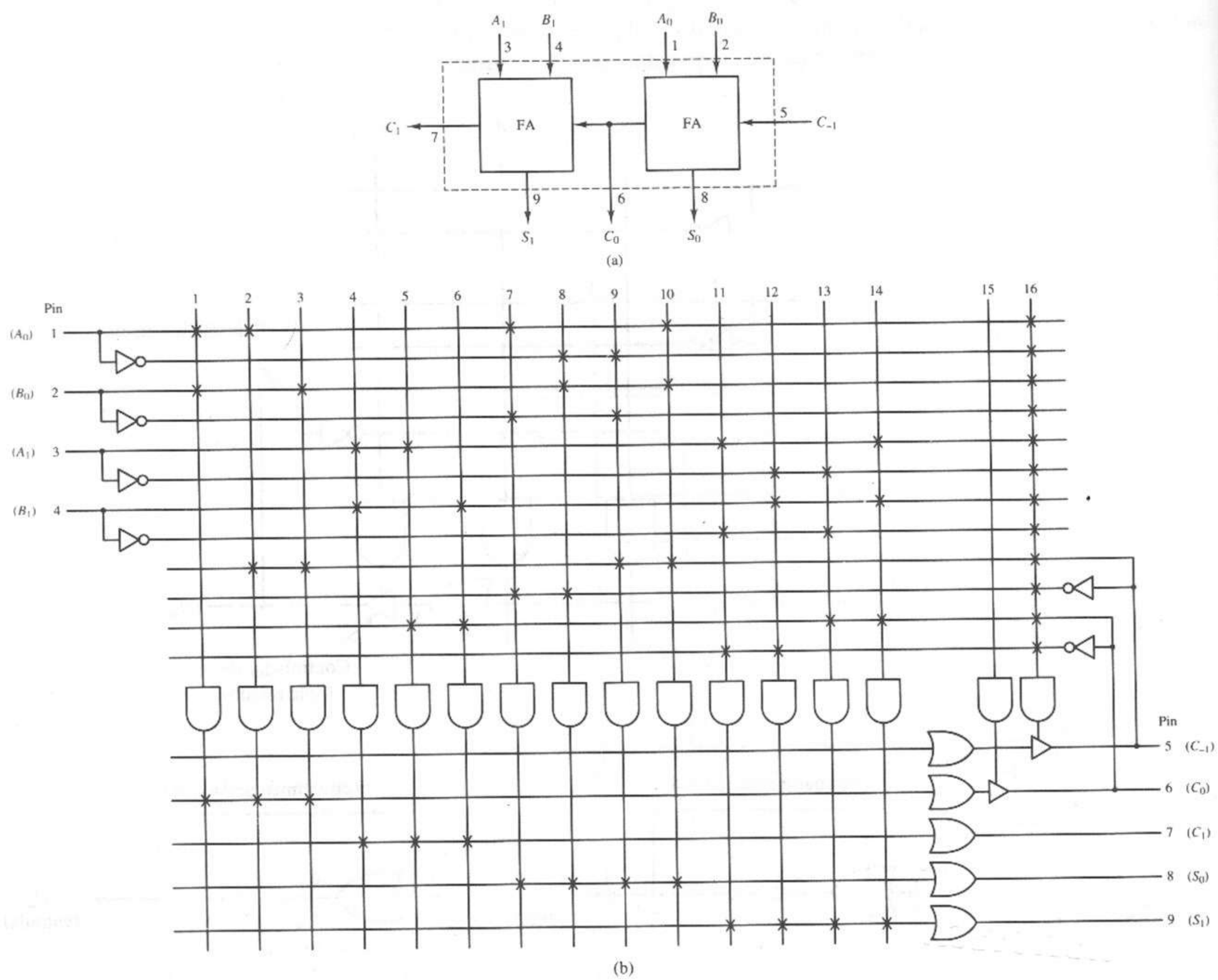


Figura 5.14 Sumador con acarreo en cascada de dos bits, con líneas de entrada/salida y retroalimentación. **(a)** Diagrama de bloques con números de *pines*. **(b)** Realización lógica programable.

en la figura 5.13c. Esto desconecta el término suma del *pin* y, mediante la línea de retroalimentación, se convierte en una entrada al arreglo AND. El funcionamiento de estos *pines* bidireccionales y de las líneas de retroalimentación se ilustra en el siguiente ejemplo.

EJEMPLO 5.3

Implantar un sumador con acarreo en cascada de 2 bits, como se muestra en la figura 5.14a, empleando un arreglo lógico programable con cuatro *pines* exclusivos para entrada, tres *pines* exclusivos para salida y dos *pines* bidireccionales.

Como vimos en el capítulo 4, las ecuaciones lógicas estándar para una etapa, i, de un sumador completo de n bits son las siguientes:

$$S_i = A_i\bar{B}_i\bar{C}_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + \bar{A}_i\bar{B}_iC_{i-1} + A_iB_iC_{i-1}$$
$$C_i = A_iB_i + A_iC_{i-1} + B_iC_{i-1}$$

donde A_i y B_i son entradas de datos y C_{i-1} la entrada de acarreo de la etapa i, S_i es la salida de suma y C_i la salida de acarreo. En un sumador con acarreo en cascada, el acarreo de salida de una etapa se conecta a la entrada de acarreo de la siguiente etapa, como se muestra en la figura 5.14a.

La figura 5.14b muestra la implantación PLA del diagrama de bloques de la figura 5.14a. Como el sumador necesita cinco entradas y sólo hay cuatro *pines* exclusivos para entrada, utilizamos el *pin* bidireccional 5 como otra entrada. Así, el controlador del *pin* 5 es desactivado por la línea de producto 16 dejando intactos todos sus fusibles. Observe que la línea de producto 16 debe tener el valor 0 ya que es el producto de todas las entradas y de sus complementos. Podríamos haber utilizado cualquier par de variables, pero al quedar intactas todas, el dispositivo es más confiable.

Nos valemos del término de acarreo C_0 para calcular los términos S_1 y C_1 a través de la línea de retroalimentación desde el *pin* 6, permitiendo que C_0 se combine con A_1 y B_1 según las ecuaciones anteriores.

5.2.7 Dispositivos comerciales

Casi todos los dispositivos lógicos programables comerciales están organizados como se muestra en la figura 5.15, de modo que las entradas se aplican a un arreglo AND en forma complementada y no complementada y las salidas del arreglo AND se aplican a un arreglo OR, lo que permite realizar varias expresiones suma de productos. Podemos controlar las salidas del arreglo OR de manera opcional para obtener una polaridad en particular. A fin de reducir la complejidad y el costo del dispositivo y aumentar la velocidad, podemos omitir los fusibles del arreglo AND o del arreglo OR, dejando ese arreglo con una configuración fija. Cuando el arreglo AND está fijo, sólo se dispone de las combinaciones de entradas provistas (los términos producto). Cuando el arreglo OR está fijo, cada salida es una suma de los términos producto seleccionados. Los dispositivos con arreglos AND fijos y arreglos OR programables se conocen como *memorias exclusivas para lectura programables* (PROM), mientras que los dispositivos con arreglos AND programables y arreglos OR fijos son los dispositivos *de arreglos lógicos programables* (PAL). Los dispositivos en los que ambos arreglos son programables son los *arreglos lógicos programables*

mediante campos (FPLA). En las siguientes secciones examinaremos estas tres configuraciones de dispositivos programables y presentaremos algunos ejemplos típicos de dispositivos estándar. Los dispositivos que examinaremos están enumerados en la tabla 5.2.

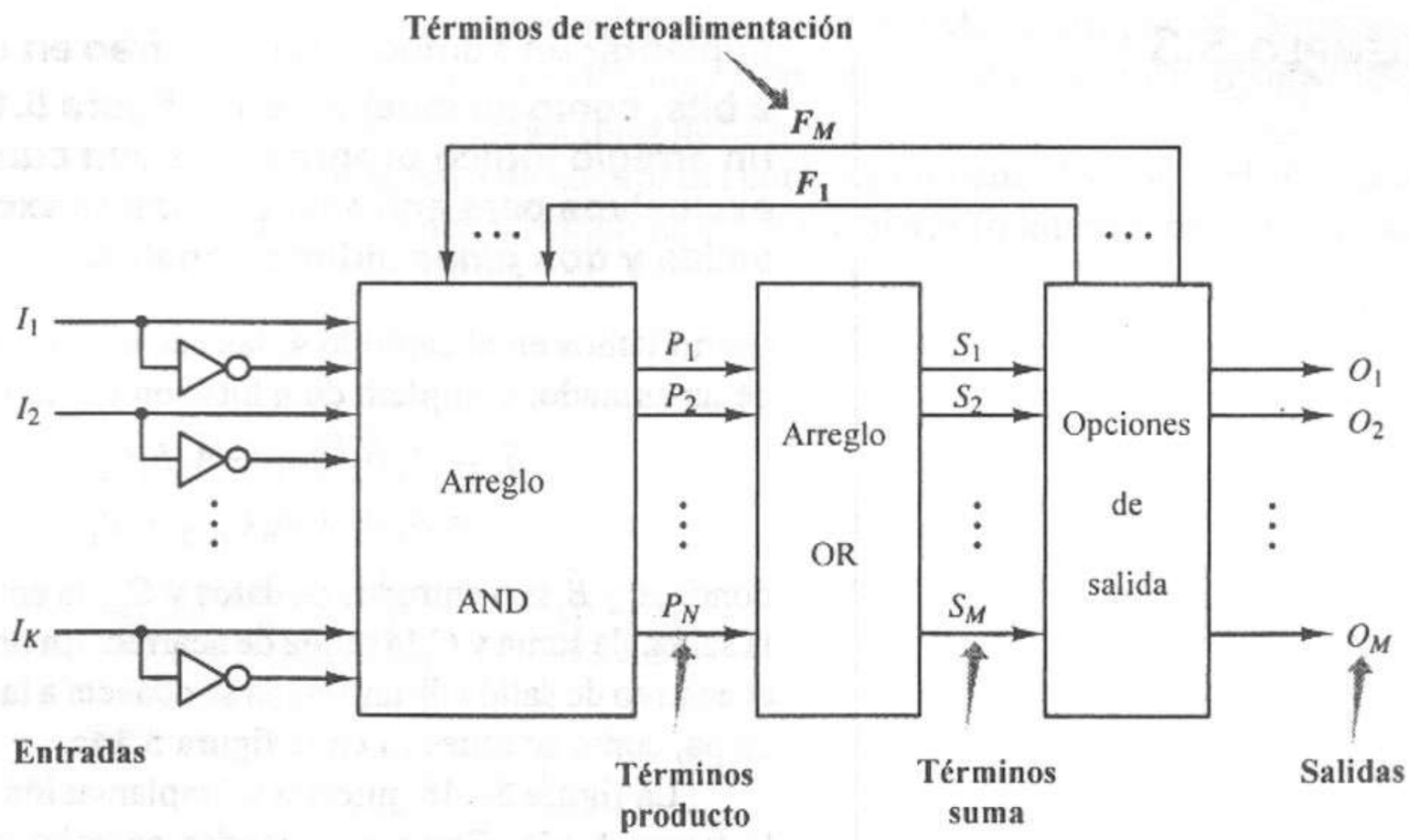


Figura 5.15 Organización básica de un dispositivo programable.

TABLA 5.2 PLD TÍPICOS DISPONIBLES EN EL COMERCIO

Dispositivo	Descripción	Entradas	Términos producto	Salidas	Polaridad de salida
PLS100	PLA	16	48	8	Programable
PLS153	PLA	16	42	10	Programable
82S123	PROM	5	32	8	Alta activa
82S129	PROM	8	256	4	Alta activa
82S131	PROM	9	512	4	Alta activa
82S135	PROM	8	256	8	Alta activa
82S137	PROM	10	1024	4	Alta activa
82S147	PROM	9	512	8	Alta activa
82S181	PROM	10	1024	8	Alta activa
82S185	PROM	11	2048	4	Alta activa
82S191	PROM	11	2048	8	Alta activa
82S321	PROM	12	4096	8	Alta activa
PAL16L8	PAL	16	8	8	Baja activa
PAL14H4	PAL	14	4	4	Alta activa
PAL16C1	PAL	16	16	1	Complementaria
PAL18P8	PAL	18	8	8	Programable

5.3 Arreglos lógicos programables mediante campos

5.3.1 Estructuras de circuitos FPLA

Los arreglos lógicos programables mediante campos (FPLA) son componentes PLA empacados que contienen fusibles en serie con cada diodo de los arreglos AND y OR, fusibles que puede eliminar el usuario. Signetics Corporation (ahora Philips Semiconductors) introdujo los primeros dispositivos FPLA estándar en 1975, incluido al PLS100 [4], que aparece en la figura 5.16, seguido por otros componentes, como el PLS153 [4], que se muestra en la figura 5.17. Estos componentes difieren en el número de entradas, términos producto y salidas disponibles, y también en la disponibilidad de polaridad de salida programable, retroalimentación y otras características.

La configuración de un FPLA es por lo general $i \times p \times o$, donde i es el número de entradas del arreglo AND, p es el número de términos producto generados en el arreglo AND y o es el número de salidas del arreglo OR. El complemento de cada entrada también se proporciona al arreglo AND, con un total de $2i$ entradas que podemos utilizar en los términos producto. En muchos dispositivos FPLA, las salidas también se retroalimentan al arreglo AND (complementadas y no complementadas), lo que hace un total de $i + o$ variables y sus complementos disponibles para crear términos producto.

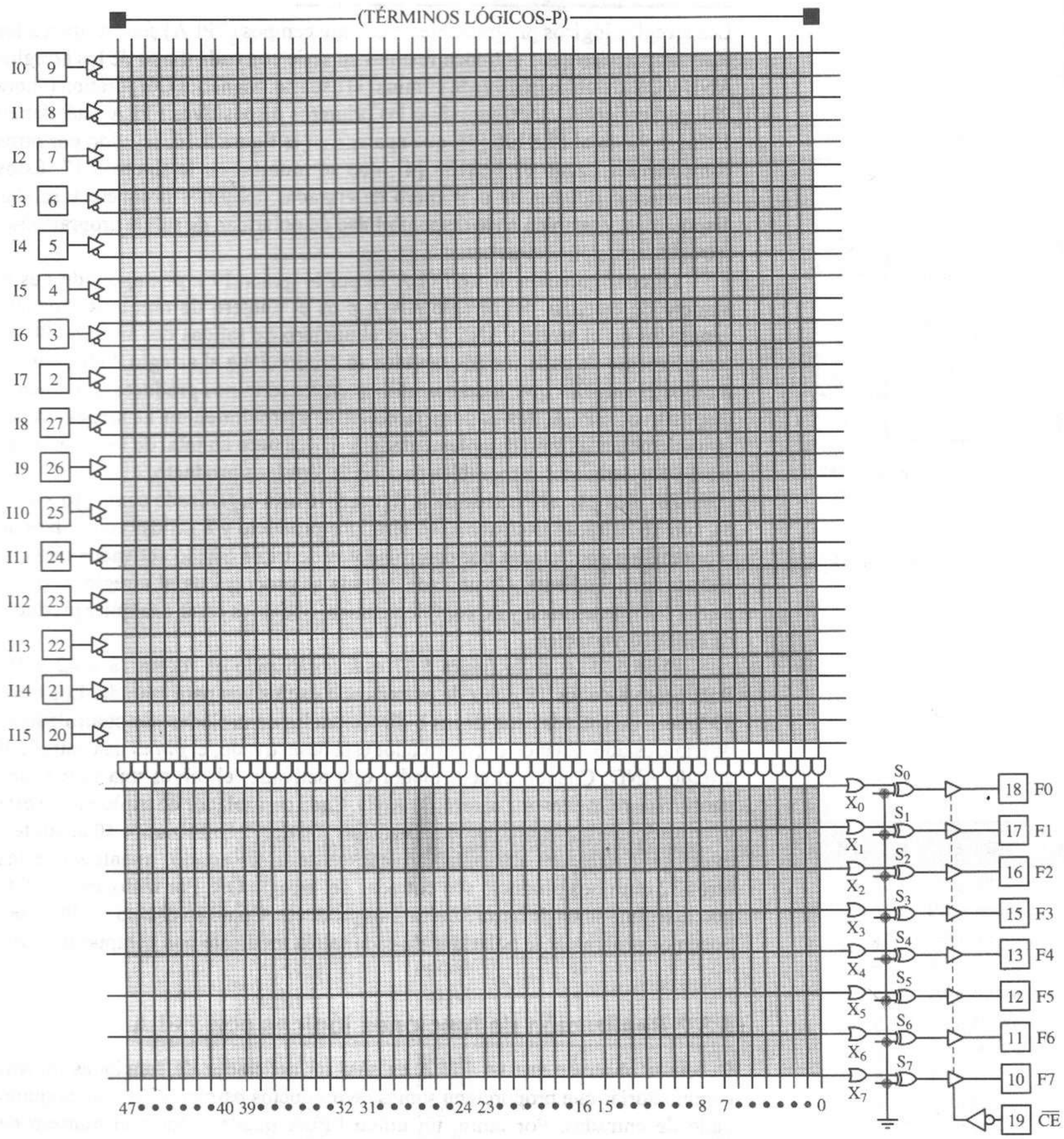
Por ejemplo, el PLS100 de la figura 5.16 está organizado como ($16 \times 48 \times 8$), con 16 entradas exclusivas, 48 términos producto y 8 salidas con polaridad programable controladas por controladores de triple estado activados con una entrada independiente. Como cada entrada se proporciona al arreglo AND en forma complementada y no complementada, podemos crear cualquier producto de hasta 16 variables.

El PLS153A de la figura 5.17 está organizado en forma $18 \times 42 \times 10$. Solamente 8 de los 18 *pines* de entrada son exclusivos para entrada. Los otros 10 "*pines* de entrada" son líneas bidireccionales controladas por controladores de triple estado. Cuando el controlador está desactivado, el *pin* es una entrada al arreglo AND. Cuando el controlador está activado, el *pin* es una salida, que también se retroalimenta al arreglo AND. Cada controlador de triple estado está controlado por un término producto independiente. Así, utilizamos 10 de los términos producto para activar controladores de triple estado, mientras que los otros 32 términos producto son entradas del arreglo OR. Por tanto, cada salida puede ser una suma de hasta 32 términos producto. Como en el caso del PLS100, podemos programar la polaridad de cada salida mediante una compuerta XOR.

5.3.2 Realización de funciones lógicas con FPLA

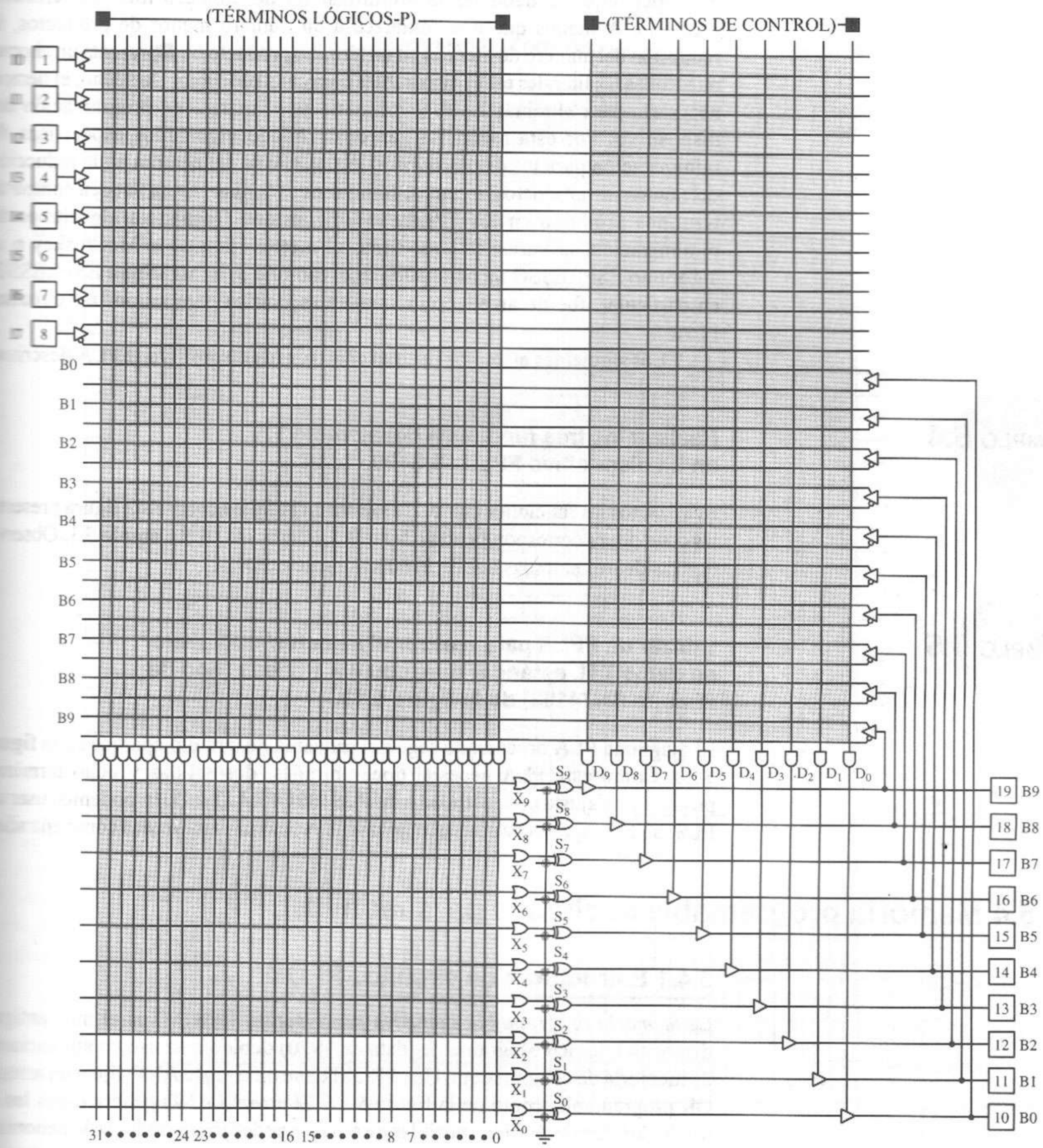
Debemos observar que un FPLA es sólo un generador de funciones lógicas combinatorias que proporciona sumas de productos parciales para un conjunto dado de entradas. Por tanto, un único FPLA puede reducir el número de componentes totales de un diseño realizando diversas funciones lógicas en un solo paquete.

Al diseñar expresiones lógicas para su implantación en un FPLA, debemos observar que podemos utilizar cualesquier literales de entrada en cada término producto y podemos incluir cualesquier términos producto en cada término suma. La principal restricción es el número total de términos producto disponibles.



NOTAS:
1. Todas las entradas de las compuertas AND con un fusible eliminado asumen el valor "1" lógico.
2. Todas las entradas de las compuertas OR con un fusible eliminado asumen el valor "0" lógico.
3. ● Conexión programable.

Figura 5.16 FPLA PLS100/101 de Philips [4]. *Fuente*: Philips, "Programmable Logic Devices (PLD) Data Handbook", Philips Semiconductor, Sunnyvale, CA, 1994.



NOTAS:
1. Todas las posiciones programadas de la compuerta 'AND' asumen el valor "1" lógico.
2. Todas las posiciones programadas de la compuerta 'OR' asumen el valor "0" lógico.
3. Conexión programable.

Figura 5.17 FPLA PLS153A de Philips [4]. *Fuente*: Philips, "Programmable Logic Devices (PLD) Data Handbook", Philips Semiconductor, Sunnyvale, CA, 1994.

Así, cuando minimizamos expresiones lógicas para realizarlas en un FPLA, el principal objetivo debe ser la minimización del número total de términos producto. A menos que nos conduzca a un número menor de productos, la reducción del número de literales en un término producto no representa un ahorro, pues todas las literales están disponibles para cada producto. De hecho, el tiempo necesario para eliminar los fusibles aumenta el tiempo de programación del dispositivo. Por esta razón, se han desarrollado algoritmos específicos de minimización para los dispositivos PLA, los que se concentran en la reducción del número de productos. Además, como varios términos suma pueden compartir términos producto en los FPLA, se utilizan con frecuencia algoritmos de minimización de varias salidas, como el método de Quine-McCluskey o el algoritmo ESPRESSO mencionados en el capítulo 3. El algoritmo ESPRESSO, en particular, fue desarrollado de manera específica para minimizar circuitos PLA.

Los siguientes ejemplos ilustran el uso de los dispositivos FPLA descritos.

EJEMPLO 5.4

Realizar las tres funciones del ejemplo 5.1 en un dispositivo FPLA PLS100.

Realizamos las funciones como se muestra en la figura 5.18. Esta figura presenta las conexiones correspondientes a la tabla 5.1 deducida en el ejemplo 5.1. Observe que las demás entradas del PLS100 son prescindibles.

EJEMPLO 5.5

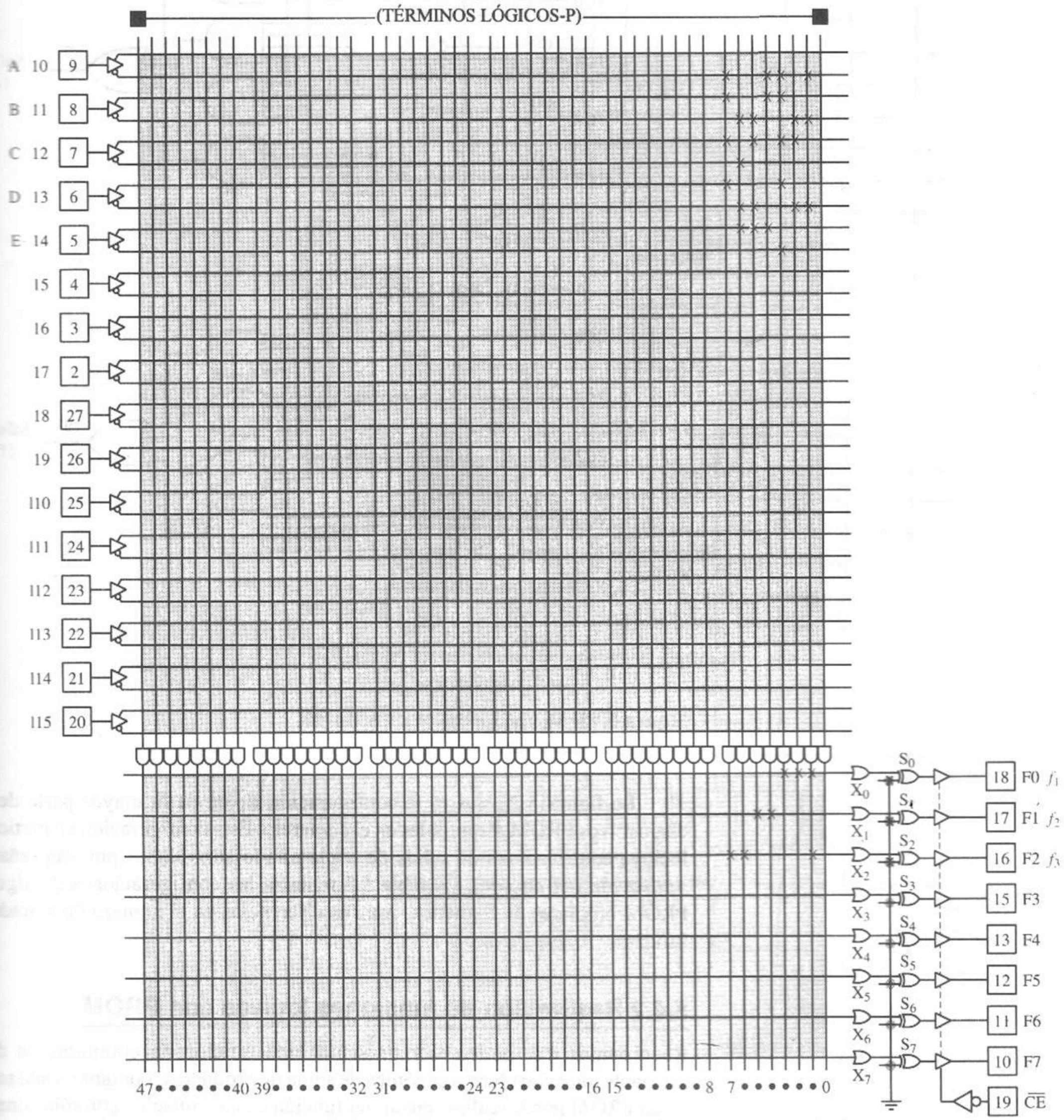
Utilizar un FPLA para realizar el selector/multiplexor de datos TTL estándar dual con 4 a 1 líneas (tipo de circuito SN74153) de la figura 5.19.

El diagrama PLA organizado según el esquema de circuito aparece en la figura 5.20. Como este PLA necesita doce entradas, dos salidas y ocho términos producto, se ajusta con facilidad a un PLS100 FPLA. También podemos usar un PLS153 FPLA, con cuatro de sus líneas bidireccionales utilizadas como entradas.

5.4 Memoria programable exclusiva para lectura

5.4.1 Estructuras de circuitos PROM

La *memoria programable exclusiva para lectura* (PROM) es el más antiguo dispositivo lógico programable (data de 1970) debido a su uso en aplicaciones de memoria de computadora. Una PROM tiene un arreglo AND fijo y un arreglo OR programable, como en la figura 5.21. El arreglo AND genera todos los 2^n productos de mintérminos posibles de sus n entradas y, por tanto, se le denomina con frecuencia *decodificador n a* 2^n (describimos los decodificadores en el capítulo 4). El arreglo OR permite incluir cualquier combinación de términos producto en cada término suma. Por tanto, podemos realizar la forma canónica de suma de productos de cualquier función directamente a partir de su tabla de verdad o su lista de mintérminos. El número de términos suma varía entre los dispositivos, según el tamaño del circuito, el número de *pines* en el paquete y otras consideraciones de costo. Los dispositivos PROM no incluyen por lo general opciones de polaridad de salida o retroalimentación.



NOTAS:
1. Todas las entradas de las compuertas AND con un fusible eliminado asumen el valor "1" lógico.
2. Todas las entradas de las compuertas OR con un fusible eliminado asumen el valor "0" lógico.
3. Conexión programable.

Figura 5.18 Realización de la tabla 5.1 con un FPLA PLS100 de Philips [4]. *Fuente*: Philips, "Programmable Logic Devices (PLD) Data Handbook", Philips Semiconductor, Sunnyvale, CA, 1994.

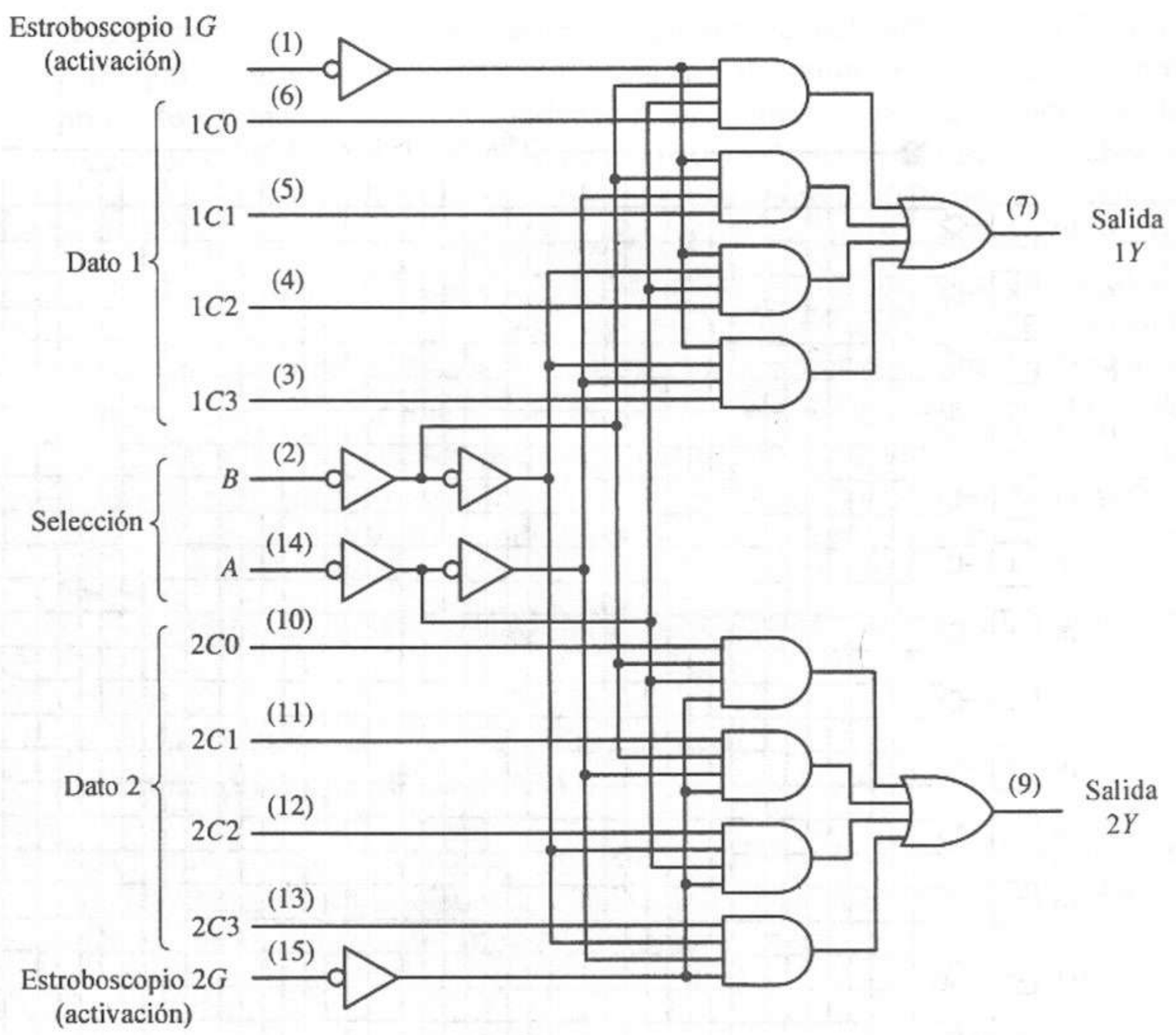


Figura 5.19 Multiplexor dual 4 a 1 SN74153.

La figura 5.22 ilustra la configuración típica de la mayor parte de los dispositivos PROM disponibles en el comercio. Esta configuración en particular incluye controladores de salida de triple estado controlados por una señal de *Activación del circuito*. La tabla 5.2 muestra las configuraciones de algunos PROM bipolares de Signetics, con sus diferencias en el número de entradas y salidas.

5.4.2 Realización de funciones lógicas con PROM

Recordemos que podemos representar una función de conmutación dada mediante una única forma canónica de suma de productos. Por tanto, cada salida de un PROM puede realizar cualquier función de conmutación con sólo conectar esa salida a los mintérminos de la función. Así, para realizar una función de conmutación dada con un PROM, primero debemos expresar la función en forma canónica de suma de productos o deducir la tabla de verdad de la función. Después, conectamos cada uno de los mintérminos de la función al término OR deseado para obtener la forma canónica SOP. Observe que *no obtenemos ventajas* minimizando la función al utilizar una PROM, ya que debemos utilizar su forma canónica para generar el mapa de fusibles PROM. También debemos señalar

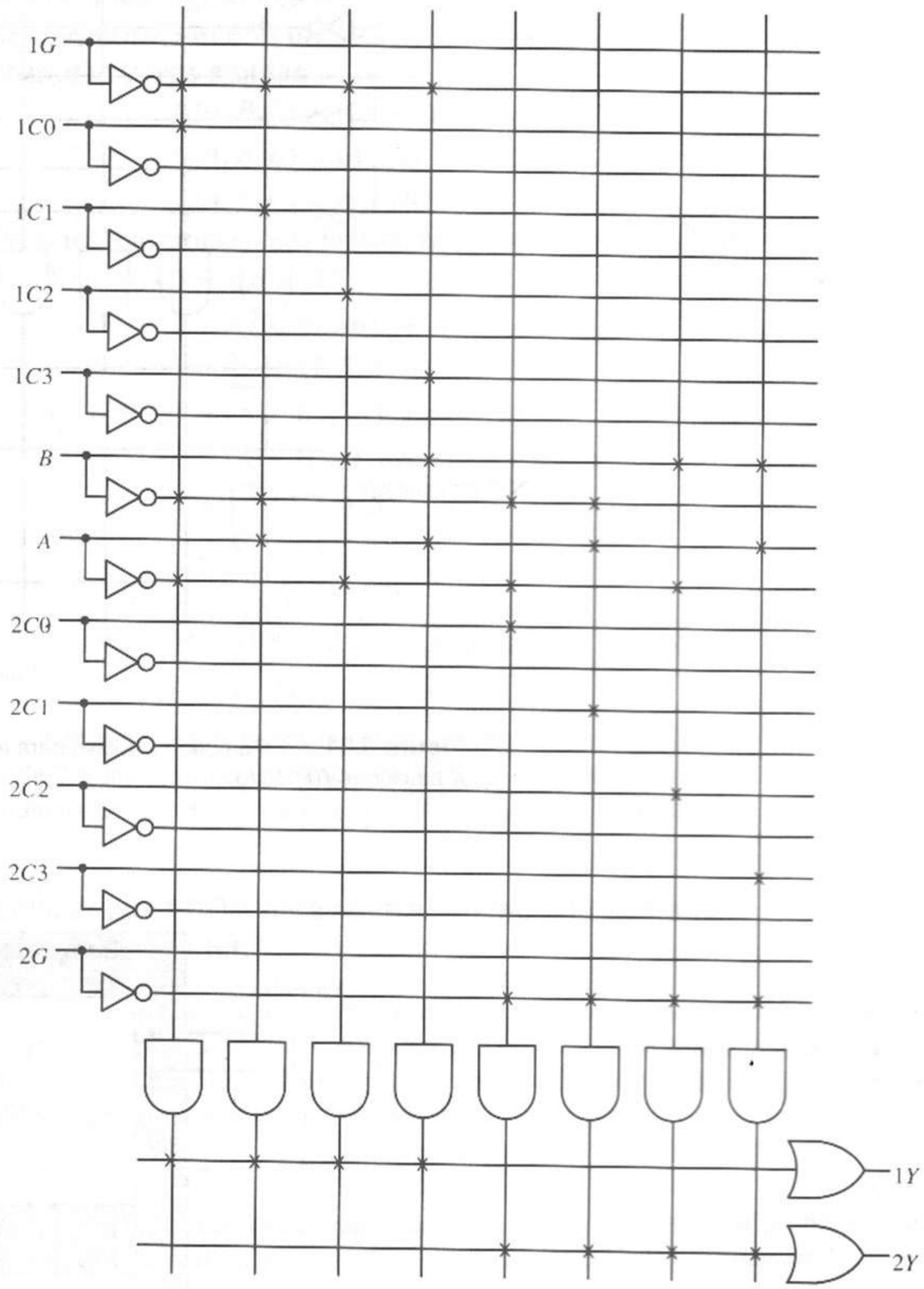


Figura 5.20 Realización PLA del multiplexor dual 4 a 1 SN74153.

que el uso de las PROM comerciales es muy ineficiente si sólo necesitamos pocos mintérminos, a menos que el principal objetivo sea minimizar el número de circuitos.

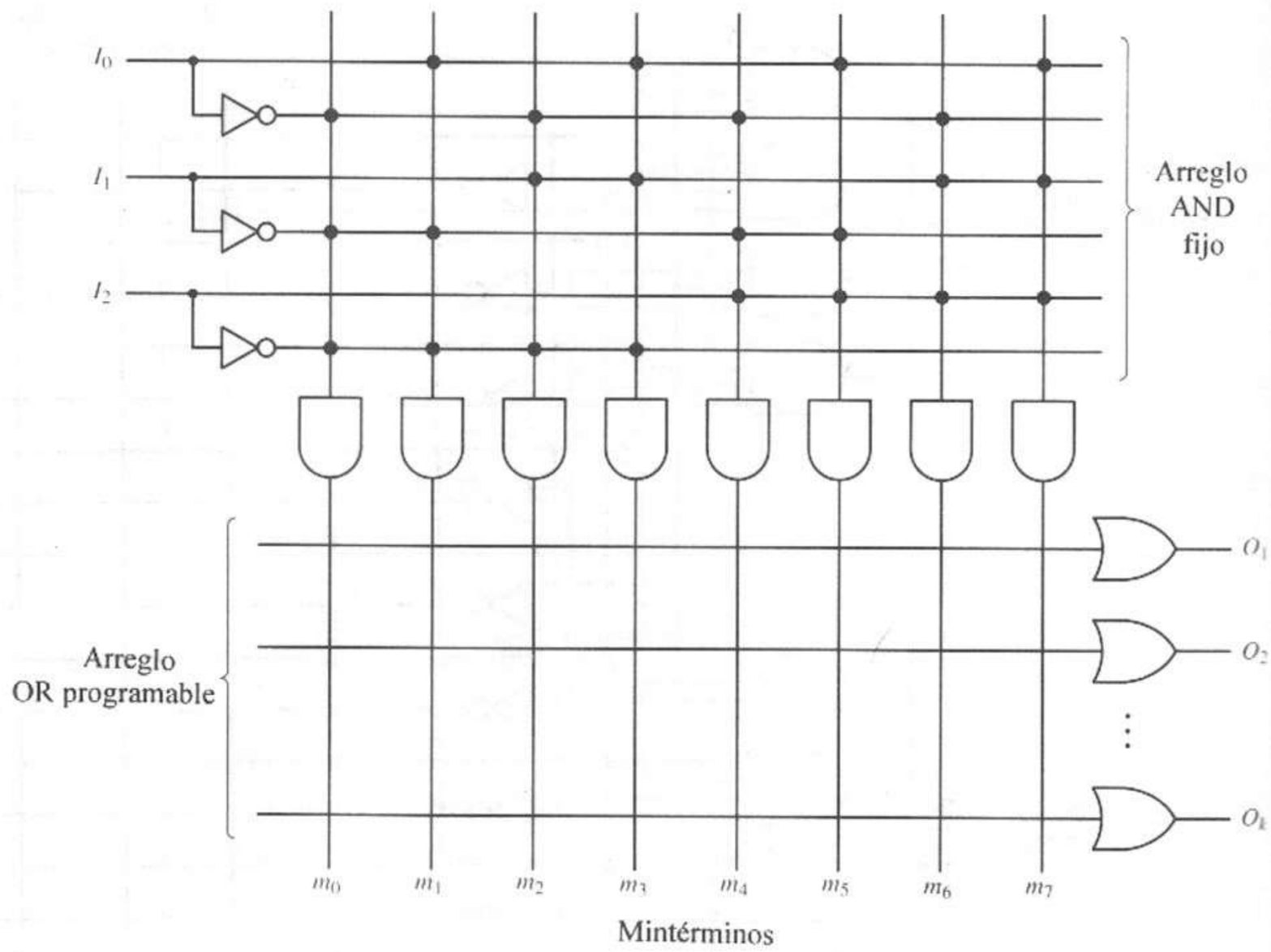


Figura 5.21 La memoria exclusiva para lectura programable (PROM) puede realizar K funciones $f(I_2, I_1, I_0)$.

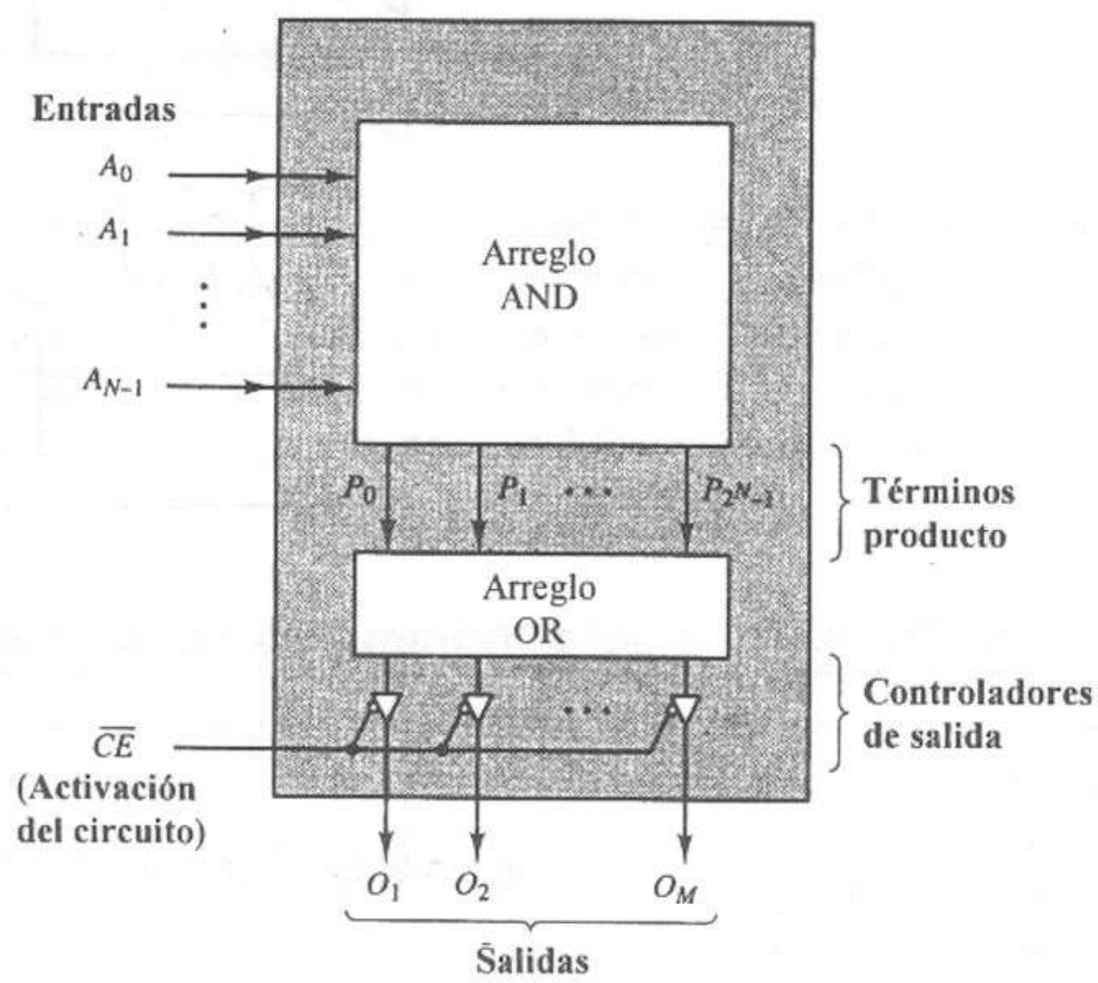


Figura 5.22 Configuración general de una PROM comercial.

EJEMPLO 5.6

Realizar las siguientes tres funciones de conmutación con una PROM de tres entradas y tres salidas.

$$f_1(A,B,C) = AB + \bar{B}C$$
$$f_2(A,B,C) = (A + \bar{B} + C)(\bar{A} + B)$$
$$f_3(A,B,C) = A + BC$$

Primero, convertimos cada función en su forma canónica SOP.

$$\begin{aligned} f_1(A, B, C) &= AB + \bar{B}C \\ &= AB\bar{C} + ABC + \bar{A}\bar{B}C + A\bar{B}C \\ &= \sum m(1, 5, 6, 7) \\ f_2(A, B, C) &= (A + \bar{B} + C)(\bar{A} + B) \\ &= (A + \bar{B} + C)(\bar{A} + B + \bar{C})(\bar{A} + B + C) \\ &= \prod M(2, 4, 5) \\ &= \sum m(0, 1, 3, 6, 7) \\ f_3(A, B, C) &= A + BC \\ &= A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC + \bar{A}BC \\ &= \sum m(3, 4, 5, 6, 7) \end{aligned}$$

Por tanto, conectamos la salida 1 a los términos producto (1, 5, 6, 7), la salida 2 a los términos producto (0, 1, 3, 6, 7) y la salida 3 con los términos producto (3, 4, 5, 6, 7). La figura 5.23 muestra el circuito final.

EJEMPLO 5.7

Utilizar una PROM para realizar un módulo sumador completo de 1 bit.

La tabla 5.3 es la tabla de verdad de un sumador completo. Utilizamos esta tabla para programar la PROM, eliminando los fusibles correspondientes a cada cero en las dos funciones, como en la figura 5.24. Observe que para este circuito necesitamos una PROM de tres entradas y dos salidas.

Las PROM son soluciones particularmente eficientes en problemas que requieren el uso de la mayor parte de los mintérminos de una función. Ejemplos de este tipo de aplicaciones son los convertidores de código, los decodificadores y las tablas de búsqueda.

EJEMPLO 5.8

Diseñar un convertidor de código binario a código Gray mediante un PROM de cuatro entradas y cuatro salidas.

La tabla de verdad del convertidor de código aparece en la tabla 5.4. Observe que en la salida aparecen 15 de los 16 mintérminos posibles (el mintérmino 0 es el único que no aparece en las cuatro salidas). La tabla de verdad se asocia a una PROM como en la figura 5.25.

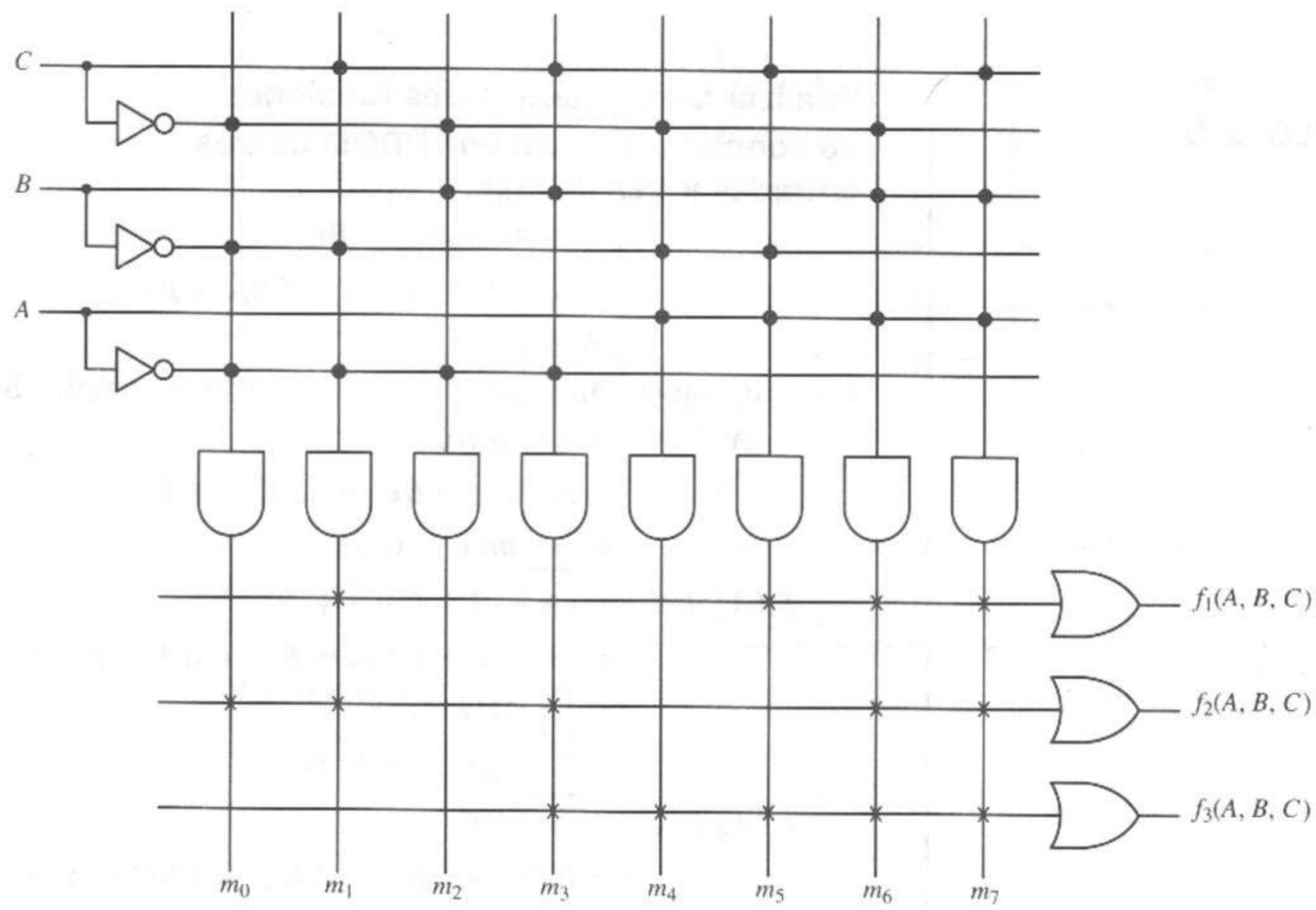


Figura 5.23 Solución PROM para el ejemplo 5.6.

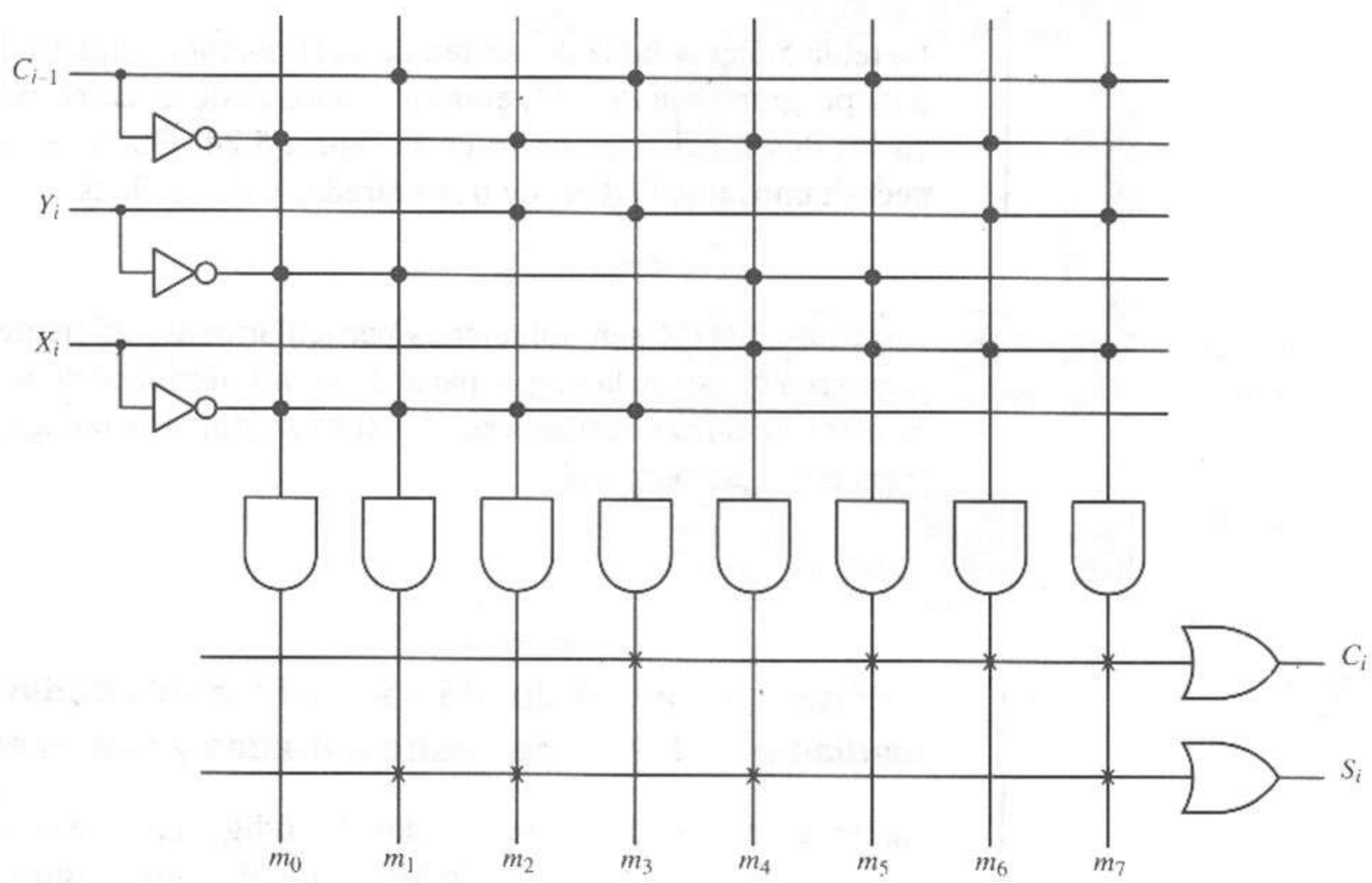


Figura 5.24 Realización PROM de un sumador completo.

TABLA 5.3 TABLA DE VERDAD DEL SUMADOR COMPLETO

X_i	Y_i	C_{i-1}	C_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

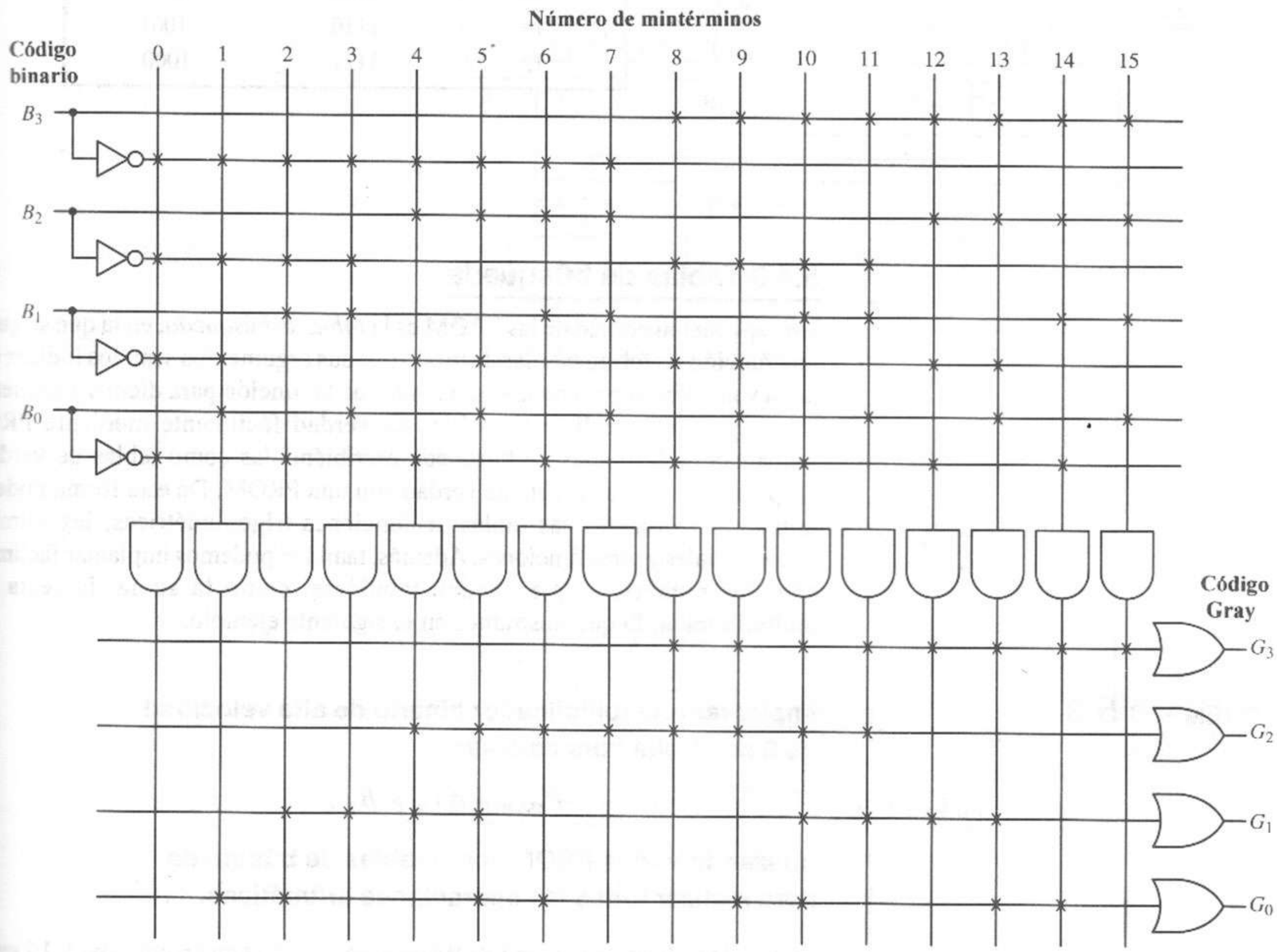


Figura 5.25 Realización PROM del convertidor de código binario a Gray.

TABLA 5.4 TABLA DE VERDAD DEL CÓDIGO BINARIO A GRAY

Número decimal	Binario $B_3B_2B_1B_0$	Código Gray $G_3G_2G_1G_0$
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

5.4.3 Tablas de búsqueda

Una aplicación común de las PROM es la *tabla de búsqueda*, en la que se guarda una función en forma tabular de modo que sus argumentos sean los índices de la tabla y nos sirvan para recuperar el valor de la función para dichos argumentos. Como podemos realizar las tablas de verdad fácilmente mediante PROM, implantamos las tablas de búsqueda escribiéndolas como tablas de verdad y después realizamos la tabla de verdad con una PROM. De esta forma podemos implantar fácilmente las tablas de funciones trigonométricas, logarítmicas, exponenciales y otras funciones. Además, también podemos implantar fácilmente cálculos numéricos que pueden tabularse, como la suma, la resta y la multiplicación, lo que ilustramos en el siguiente ejemplo.

EJEMPLO 5.9

Implantar un multiplicador binario de alta velocidad de 8 por 8 bits para calcular

$$P_{15-0} = A_{7-0} \times B_{7-0}$$

utilizando varias PROM como tablas de búsqueda para realizar todas las operaciones aritméticas.

En vez de utilizar una única PROM de gran tamaño con 16 entradas y 16 salidas para implantar una tabla de multiplicación con 2^{16} filas, separamos los dos operandos en cantidades de 4 bits como sigue.

$$
\begin{aligned}
P_{15-0} &= A_{7-0} \times B_{7-0} \\
&= ((A_{7-4} \times 2^4) + A_{3-0}) \times ((B_{7-4} \times 2^4) + B_{3-0}) \\
&= (A_{7-4} \times B_{7-4}) \times 2^8 + ((A_{7-4} \times B_{3-0}) + (A_{3-0} \times B_{7-4})) \times 2^4 \\
&\quad + A_{3-0} \times B_{3-0}
\end{aligned}
$$

Podemos efectuar esta operación con cuatro multiplicadores de 4 por 4 bits para calcular productos parciales y tres sumadores binarios para sumar los productos parciales. Podemos multiplicar por 2^4 y 2^8 simplemente recorriendo los términos correspondientes 4 y 8 bits hacia la izquierda, respectivamente. Observe que la tabla para una multiplicación de 4 por 4 bits sólo tiene 16 filas.

El diagrama de bloques de la figura 5.26 es un sistema de varios PROM utilizado para implantar el multiplicador. Programamos las PROM 1 a 4 como

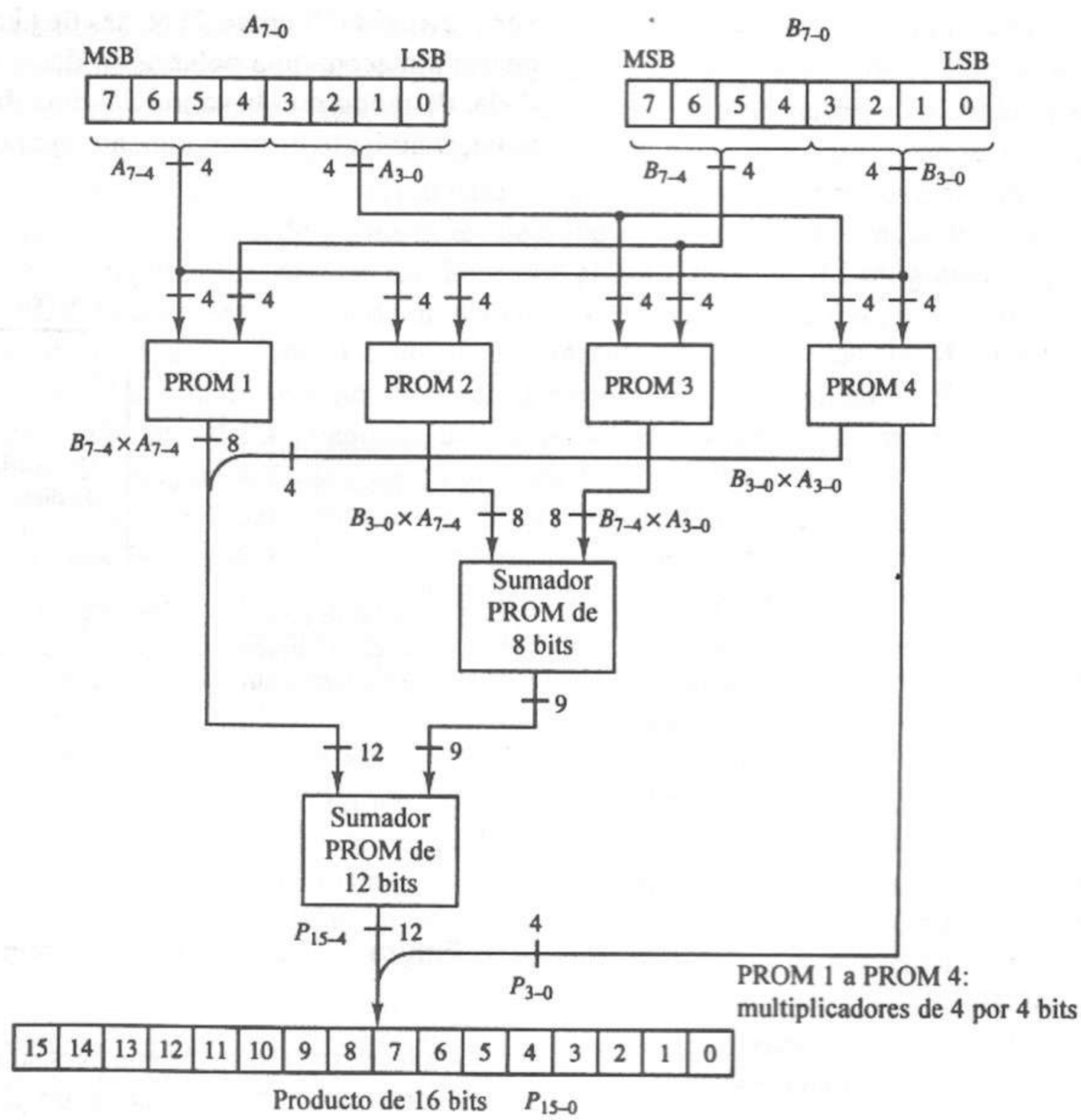


Figura 5.26 Implantación de un multiplicador binario de alta velocidad con PROMs. (Los PROM 1 a 4 son multiplicadores de 4 × 4).

tablas de búsqueda de multiplicación para formar productos parciales de 4 por 4 bits. Utilizamos sumadores para sumar los productos parciales, los cuales también se implantan mediante PROM, para formar el producto final.

5.4.4 Aplicaciones generales de la memoria exclusiva para lectura

Los dispositivos PROM se utilizan con frecuencia en las computadoras y otros sistemas digitales como memorias exclusivas para lectura (ROM) para almacenamiento permanente (no volátil) de información tal como programas de cómputo, tablas de valores constantes y tablas para traducción de código. Observemos la figura 5.22, podemos leer la información de una ROM especificando su número de fila en la tabla (su *dirección*) en las entradas $A_{n-1}...A_0$. La palabra de datos seleccionada aparece en las salidas $O_m...O_1$ después de un retardo corto, llamado *tiempo de acceso* del dispositivo.

Una PROM de n entradas y m salidas puede almacenar una tabla de hasta 2^n palabras de datos de m bits. Consideremos el diagrama PROM de la figura 5.27. El arreglo AND es de hecho un decodificador n a 2^n, donde cada salida del decodificador corresponde a un mintérmino de las entradas $A_{n-1}...A_0$. Podemos ver el arreglo OR como 2^n celdas de almacenamiento de m bits, cada una de las cuales almacena una palabra de datos de m bits. Se decodifica una dirección dada, de modo que la salida del decodificador activado seleccione la celda de almacenamiento correspondiente para controlar las salidas $O_m...O_1$.

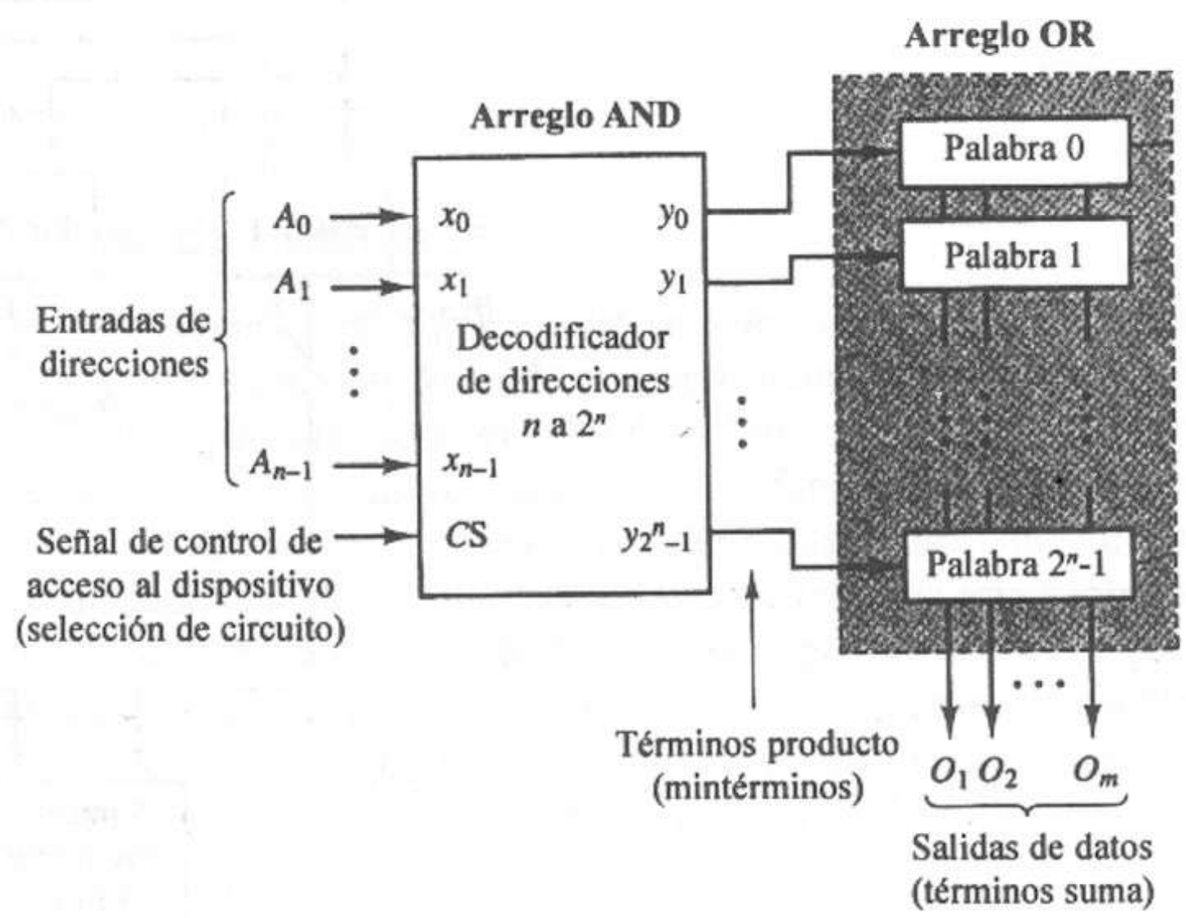


Figura 5.27 Dispositivo PROM como memoria exclusiva para lectura $2^n \times m$.

La información se ordena para su almacenamiento en una PROM determinando la dirección donde se colocará cada palabra de datos. En gran parte de los casos, la información simplemente se coloca en posiciones adyacentes, a partir de la primera dirección dentro de la PROM. Sin embargo, en algunas situaciones la información se ordena en forma de tablas de búsqueda, donde cada dirección es un código de datos específico relacionado de alguna manera con la información almacenada en la PROM.

5.4.5 Tecnologías de memorias exclusivas para lectura

Además de las PROM vinculadas mediante fusibles ya descritos, la memoria exclusiva para lectura y otros dispositivos programables están disponibles en muchas otras tecnologías, con diversos niveles de costo, velocidad, flexibilidad y reutilizabilidad.

La complejidad de un dispositivo PROM está determinada por el número de diodos y fusibles que contiene. Una PROM de n entradas tiene $2n$ diodos que conectan las entradas a cada término producto (n variables no complementadas y n complementadas). Como pueden existir 2^n términos producto de n variables, el arreglo AND incluye $2n \times 2^n$ diodos. Si hay k salidas, existen $k \times 2^n$ diodos y fusibles en el arreglo OR, ya que podemos conectar cada uno de los 2^n términos producto a cada salida. Por tanto, el costo total es $(2n + k) \times 2^n$ diodos más $k \times 2^n$ fusibles.

Para aplicaciones de gran volumen se utilizan por lo general *memorias exclusivas para lectura programables mediante plantilla*, o simplemente ROM. En los dispositivos ROM, no hay fusibles programables por el usuario. En vez de eso, durante los pasos finales del proceso de fabricación del circuito en la fábrica el arreglo OR se configura de manera permanente colocando u omitiendo alambres que están en serie con los diodos para representar los fusibles intactos o eliminados, respectivamente. Una *plantilla* adaptada indica dónde colocar los alambres durante esta etapa de fabricación y, por tanto, determina las celdas que serán 1 y las que serán 0. Cada plantilla ROM se adapta a partir de una tabla suministrada por el cliente y, por tanto, tiene un costo de desarrollo relativamente alto, por lo general de varios miles de dólares. Aunque la ausencia de fusibles programables en el arreglo OR hace que el costo de la ROM programable por plantilla sea menor que el de un dispositivo PROM comparable, este ahorro en costo es opacado por el costo de la plantilla. Por tanto, las ROM sólo son costeables cuando se solicitan muchos dispositivos con la misma información, pero para un número pequeño de componentes, es mejor utilizar dispositivos PROM programables de manera individual por el cliente.

Durante el desarrollo de un circuito lógico, la información por almacenar en cada PROM experimenta cambios frecuentes hasta que el diseño ha sido depurado por completo. Por desgracia, las ROM y las PROM no se pueden modificar una vez programadas. Hay que desecharlas y programar nuevos dispositivos para remplazarlas. En estos caso, se utilizan por lo general *memorias programables exclusivas para lectura borrables* (EPROM). El arreglo OR de una EPROM se programa con un voltaje de programación especial que captura la carga eléctrica sólo en algunas celdas de almacenamiento. La presencia o ausencia de carga en una celda indica un 0 o un 1 lógico. Aunque no es permanente como un fusible eliminado, la carga perdura hasta 10 años. Sin embargo, se puede disipar al irradiar el circuito con luz ultravioleta a través de una ventana de cuarzo que tiene el *chip*, lo cual devuelve el arreglo OR a su condición inicial (sin programación). Podemos entonces reprogramar la EPROM con nueva información. Este ciclo de borrado y la reprogramación puede repetirse hasta que el diseño sea correcto, lo que permite utilizar una única EPROM en todo el desarrollo.

Una EEPROM (*memoria programable exclusiva para lectura borrable eléctricamente*) es similar a una EPROM en el sentido de que también representa unos y ceros en sus celdas de memoria mediante la presencia o ausencia de una carga eléctrica. Como en la EPROM, podemos disipar esta carga y reprogramar

el circuito. Sin embargo, en una EEPROM, el borrado se hace eléctricamente, aplicando un voltaje especial al circuito. Esto permite borrar y reprogramar un circuito sin sacarlo del producto. Por tanto, las EEPROM son atractivas en aplicaciones en las que hay que modificar la información sin manipular físicamente el circuito. Muchos dispositivos EEPROM permiten el borrado selectivo del circuito; es decir, permiten borrar localidades específicas sin perturbar otras. También existen dispositivos EEPROM de menor costo, llamados *memorias flash*, que sólo permiten el borrado de todo el circuito, sacrificando la flexibilidad en aras del costo.

Los dispositivos EPROM y EEPROM son más complejos que los dispositivos PROM y, por tanto, son más caros por bit que los PROM comparables. Además, la mayor parte de los EPROM y EEPROM tiene retardos por propagación mayores que los PROM y ROM de tamaño comparable, principalmente porque los primeros se fabrican con tecnologías de transistores NMOS o CMOS, mientras que los PROM y ROM utilizan por lo general TTL bipolar. Sin embargo, es frecuente que el mayor costo y el menor rendimiento sean menos importantes que la conveniencia de poder borrar y reprogramar un circuito. En algunos casos, los EPROM y EEPROM utilizados durante la etapa de elaboración del prototipo se conservan como producto final. Sin embargo, cuando hay que fabricar en masa un producto, el costo a menudo se reduce remplazando los EPROM y EEPROM por sus PROM y ROM equivalentes después de finalizar el diseño.

5.5 Lógica de arreglos programables

5.5.1 Estructuras de circuitos PAL

Monolithic Memories, Inc. introdujo los dispositivos PAL (o simplemente PAL) a finales de la década de 1970, como un sustituto de bajo costo de las compuertas lógicas discretas, las PROM y los PLA [5]. Un PAL, como se aprecia en la figura 5.28, tiene un arreglo AND programable y un arreglo OR fijo. En el arreglo OR fijo, cada línea de salida se conecta de manera permanente con un conjunto específico de términos producto. Por ejemplo, en el PAL de la figura 5.28 cada línea de salida se conecta a tres líneas de producto y, por tanto, representa una suma de tres términos producto. Debido a la presencia del arreglo OR fijo, la representación PAL de la figura 5.29 se usa con más frecuencia que la de la figura 5.28.

A diferencia de las PROM, donde se generan los 2^n productos de n variables posibles, los PAL sólo generan un número limitado de términos producto, dejando que el diseñador elija los productos generados por cada suma. Por tanto, el costo total de un PAL es mucho menor que el costo de los PROM y FPLA comparables.

Como en el caso de las memorias exclusivas para lectura, los PAL están disponibles en diversas tecnologías de circuitos, además de la TTL bipolar programable mediante fusibles. En particular, las tecnologías de EPROM y EEPROM antes descritas, que utilizan tecnologías de transistores NMOS y CMOS, se emplean con frecuencia para los arreglos AND programables de varios dispositivos PAL para poder borrarlos y reprogramarlos. Estos circuitos se conocen como *dispositivos lógicos programables borrables* o EPLD. Como en el caso de las memorias exclusivas para lectura, los EPLD con tecnología CMOS y NMOS son por lo general más costosos y tienen retardos por propagación

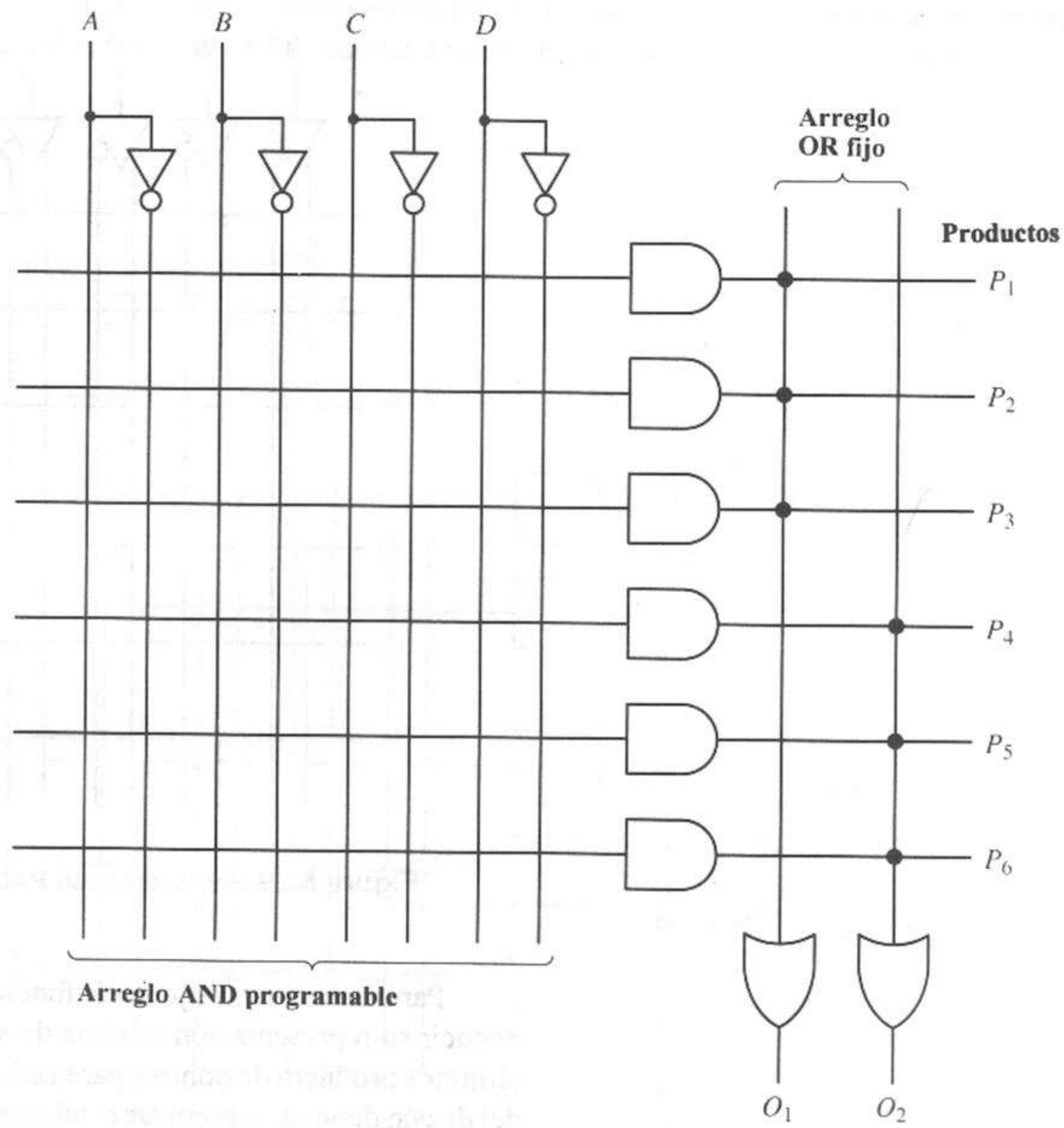


Figura 5.28 Dispositivo de arreglo lógico programable (PAL).

mayores que los PAL comparables con TTL bipolar programable mediante fusibles. Sin embargo, el hecho de poder borrar y reprogramar los circuitos hace a los EPLD más atractivos para muchas aplicaciones.

5.5.2 Realización de funciones lógicas con PAL

Como cada salida debe ser la suma de un conjunto fijo de términos producto, los PAL están más limitados que las PROM y los FPLA en cuanto al número de funciones de conmutación que pueden realizar. Por tanto, la selección de un dispositivo PAL para una aplicación particular debe garantizar que el número de términos producto por salida sea suficiente para el número de productos en el peor de los casos para esa aplicación. Otra limitación es que dos términos suma no pueden compartir un término producto individual. Si dos sumas tienen un término producto en común, hay que generar dos veces ese producto. Por fortuna, podemos representar muchas funciones de conmutación mediante sumas de una cantidad limitada de términos producto. En consecuencia, los PAL tienen mayor efectividad de costo que las PROM y FPLA para funciones que tienen muchas variables de entrada pero sólo un número pequeño de términos producto.

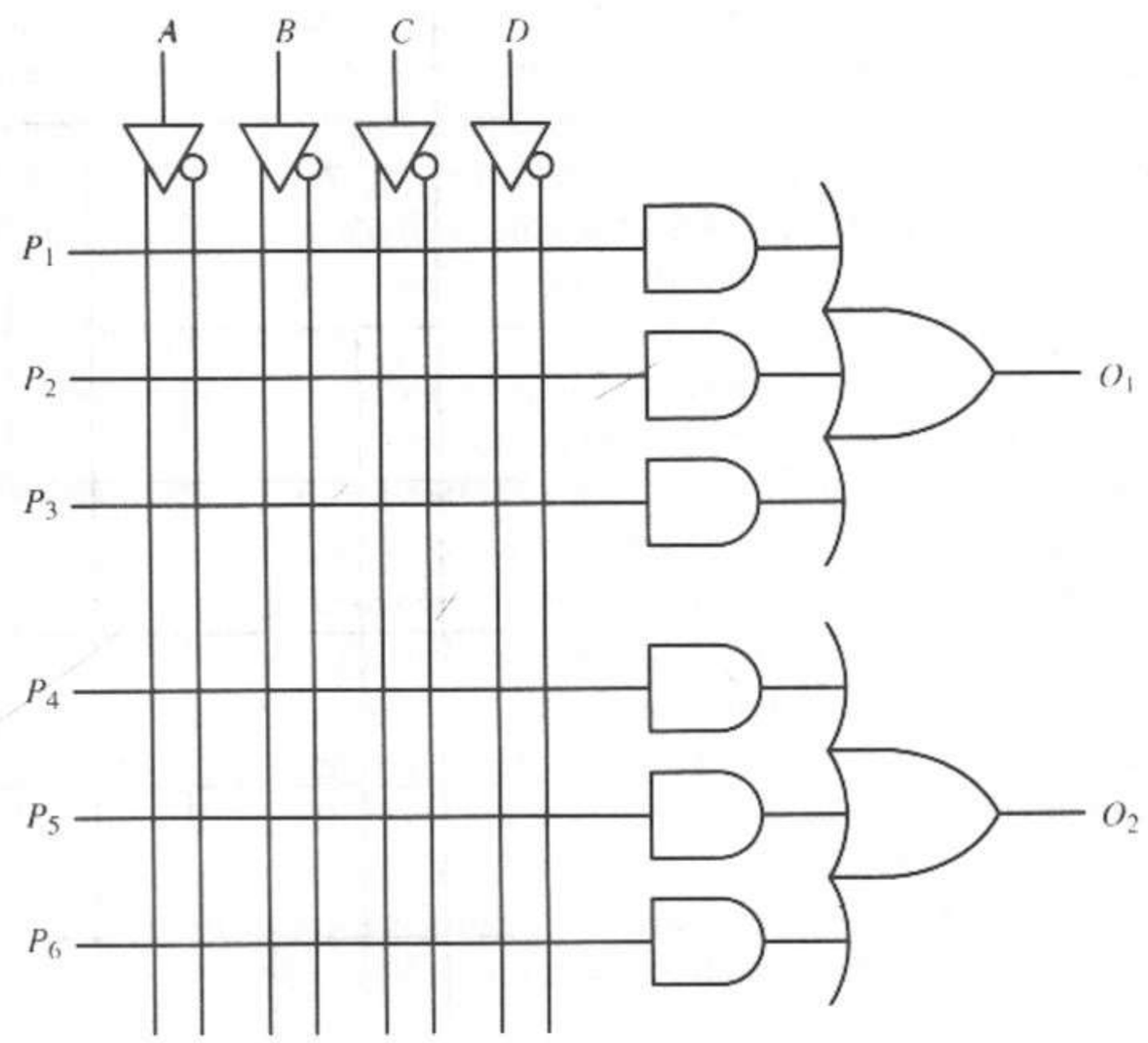


Figura 5.29 Representación PAL estándar.

Para realizar un conjunto de funciones de conmutación en un PAL, debemos deducir su representación mínima de suma de productos. Como el conjunto de términos producto disponible para cada función es limitado, el principal objetivo del diseño debe ser minimizar el número de términos producto de cada expresión SOP, en vez del número total de literales. Disponemos de cada entrada y de su complemento para cada término producto. Por tanto, no hay una ventaja real en el costo por el hecho de reducir el número de literales en cualquier término producto individual. Además, como las salidas no pueden compartir los términos producto, como en las PROM y los FPLA, no tenemos que utilizar un algoritmo de minimización de varias salidas, como el que vimos en el capítulo 3, a fin de minimizar las funciones de manera colectiva. Para realizar un PAL, debemos minimizar cada suma de manera independiente.

EJEMPLO 5.10

El ejemplo 3.24 ilustró la minimización simultánea de tres funciones:

$$f_\alpha(A, B, C, D) = \sum m(0, 2, 7, 10) + d(12, 15)$$
$$f_\beta(A, B, C, D) = \sum m(2, 4, 5) + d(6, 7, 8, 10)$$
$$f_\gamma(A, B, C, D) = \sum m(2, 7, 8) + d(0, 5, 13)$$

Obtuvimos como resultado las siguientes tres expresiones:

$$f_\alpha(A, B, C, D) = \bar{A}\bar{B}\bar{D} + \bar{B}C\bar{D} + \bar{A}BCD$$
$$f_\beta(A, B, C, D) = \bar{A}B + \bar{B}C\bar{D}$$
$$f_\gamma(A, B, C, D) = \bar{A}\bar{B}\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}BCD$$

Estas tres expresiones requieren un total de ocho términos producto, aunque sólo hay cinco términos distintos, tres de los cuales están compartidos por varias salidas. La figura 5.30 muestra una realización PAL de estas expresiones.

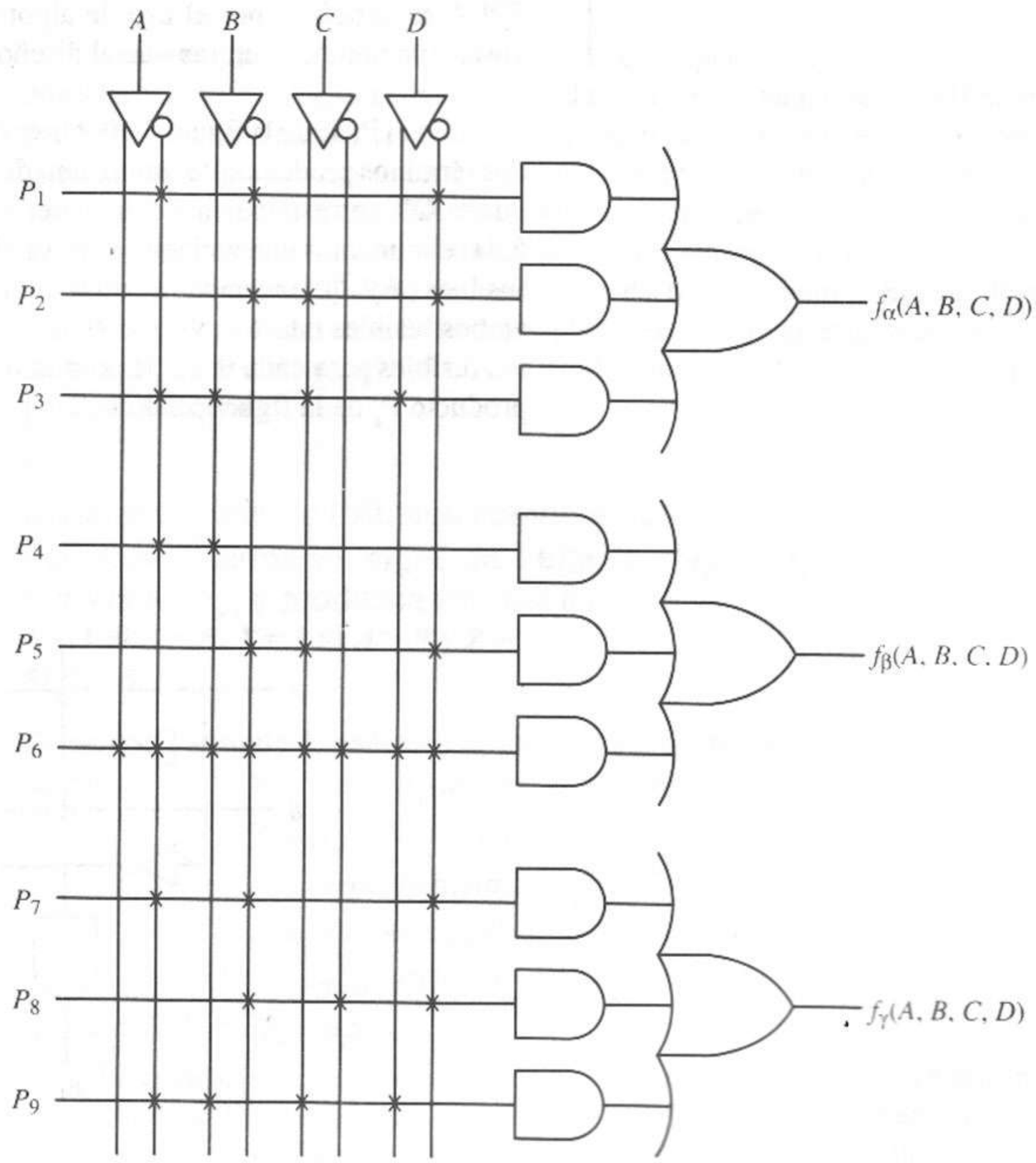


Figura 5.30 Realización PAL de $f_\alpha(A, B, C, D)$, $f_\beta(A, B, C, D)$ y $f_\gamma(A, B, C, D)$.

La minimización independiente de cada función produce las siguientes expresiones:

$$f_\alpha(A, B, C, D) = \bar{A}\bar{B}\bar{D} + \bar{B}C\bar{D} + BCD$$
$$f_\beta(A, B, C, D) = \bar{A}B + \bar{B}C\bar{D}$$
$$f_\gamma(A, B, C, D) = \bar{A}\bar{B}\bar{D} + \bar{B}\bar{C}\bar{D} + \bar{A}BD$$

Éstas también contienen un total de ocho productos, seis de ellos únicos, con dos literales menos que el conjunto anterior de expresiones.

Las realizaciones PAL de estas funciones requieren un total de cuatro entradas y ocho términos producto en cada caso, por lo que no hay un ahorro en el costo debido a las literales reducidas o los productos compartidos. Sin embargo, podemos reducir el costo de una realización FPLA utilizando el primer conjunto de expresiones, ya que sólo hay que generar cinco términos producto, en vez de los seis necesarios para el segundo conjunto de expresiones. Así, un diseño FPLA se beneficia por el uso de algoritmos que minimizan simultáneamente varias funciones, mientras que el diseño PAL no lo hace.

En el PAL de la figura 5.30, observe que la función $f_\beta(A, B, C, D)$ sólo tiene dos términos producto. Por tanto, uno de los tres productos conectados a la compuerta OR correspondiente debe tener el valor 0. Como se aprecia en la figura 5.31, eliminamos una variable, A, de un término producto (P_3) eliminando ambos fusibles (A y $\overline{A}$). Hacemos que un término producto (P_4) tenga el valor 0 dejando ambos fusibles intactos, ya que $A \cdot \overline{A} = 0$. Por lo general, dejamos intactos todos los fusibles para cada línea de producto que deba asumir el valor 0, como en el producto P_6 de la figura 5.30.

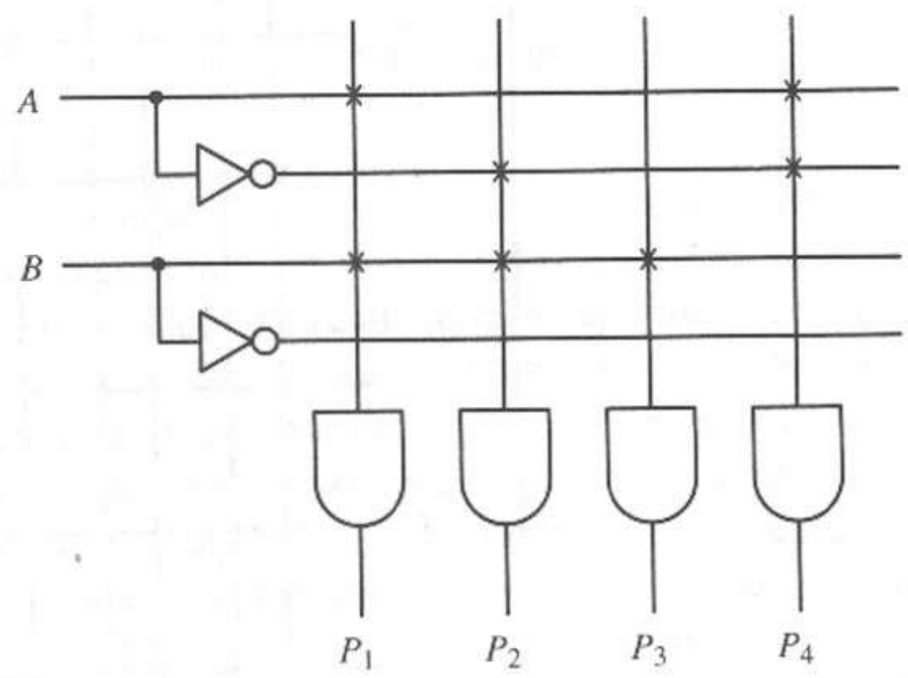


Figura 5.31 Términos producto en los que interviene la variable A y su complemento.

5.5.3 Opciones de salida y retroalimentación de PAL

Los PAL TTL estándar y CMOS se clasifican según su número de entradas, salidas, términos producto por salida y opciones de salida. Muchos dispositivos PAL incluyen opciones de polaridad de salida y retroalimentación interna de las salidas al arreglo AND. Otras características variables entre los dispositivos PAL incluyen la velocidad de conmutación y el consumo de energía.

Por lo general, la configuración de un dispositivo PAL se especifica mediante su número de componente como sigue:

PAL $I\,A\,O$

donde I es el número de entradas, O es el número de salidas y A especifica la arquitectura (polaridad de salida) de las salidas, como sigue:

A	arquitectura de salida
L	baja activa
H	alta activa
P	polaridad programable
C	salidas complementarias

Por ejemplo, el dispositivo PAL16L8 de la figura 5.32 es un PAL con 16 entradas y 8 salidas bajas activas [5]. Cada salida es la suma de siete términos producto y es controlada por un búfer de triple estado que a su vez es controlado por un término producto adicional. Seis salidas se retroalimentan al arreglo AND, y las otras dos no. El dispositivo PAL18P8, que aparece en la figura 5.33, tiene 8 *pines* bidireccionales, que incluyen salidas de polaridad programable, y 10 entradas exclusivas [5]. Los controladores de triple estado sobre las líneas bidireccionales no aparecen de forma explícita en este diagrama, sino que se incluyen en la compuerta XNOR. Otras configuraciones de dispositivo PAL se pueden encontrar en [5].

EJEMPLO 5.11

Diseñar un circuito PAL que compare dos números binarios sin signo de 4 bits, $A = (a_3a_2a_1a_0)_2$ y $B = (b_3b_2b_1b_0)_2$ y produzca tres salidas: $X = 1$ si $A = B$, $Y = 1$ si $A > B$ y $Z = 1$ si $A < B$.

Como en el capítulo 4, podemos deducir las siguientes ecuaciones para las tres salidas.

$$X = (a_3 \odot b_3)(a_2 \odot b_2)(a_1 \odot b_1)(a_0 \odot b_0)$$
$$Y = a_3\bar{b}_3 + (a_3 \odot b_3)a_2\bar{b}_2 + (a_3 \odot b_3)(a_2 \odot b_2)a_1\bar{b}_1 + (a_3 \odot b_3)(a_2 \odot b_2)(a_1 \odot b_1)a_0\bar{b}_0$$
$$Z = \bar{a}_3b_3 + (a_3 \odot b_3)\bar{a}_2b_2 + (a_3 \odot b_3)(a_2 \odot b_2)\bar{a}_1b_1 + (a_3 \odot b_3)(a_2 \odot b_2)(a_1 \odot b_1)\bar{a}_0b_0$$

donde $a_i \odot b_i = \bar{a}_i\bar{b}_i + a_ib_i$.

El desarrollo de estas ecuaciones en su forma SOP produce 16 términos producto para X, 15 productos para Y y 15 para Z. Como el número de términos producto disponibles para las sumas de los dispositivos PAL típicos por lo general es mucho menor, preferimos generar los cuatro términos

$$E_i = a_i \odot b_i, \quad \text{para } i = 0, 3$$

y retroalimentar estos términos al arreglo AND. El conjunto de ecuaciones es entonces

$$X = E_3E_2E_1E_0$$
$$Y = a_3\bar{b}_3 + E_3a_2\bar{b}_2 + E_3E_2a_1\bar{b}_1 + E_3E_2E_1a_0\bar{b}_0$$
$$Z = \bar{a}_3b_3 + E_3\bar{a}_2b_2 + E_3E_2\bar{a}_1b_1 + E_3E_2E_1\bar{a}_0b_0$$

El peor caso es ahora cuatro términos producto por salida. Esto cabe de manera conveniente en un dispositivo PAL18P8, como en la figura 5.34, con ocho de las diez entradas exclusivas para A y B, cuatro de las ocho salidas para los términos de retroalimentación E_i y tres de las salidas para X, Y y Z. Observe que se dispone de ocho términos producto por salida, más que suficiente para la función de comparación.

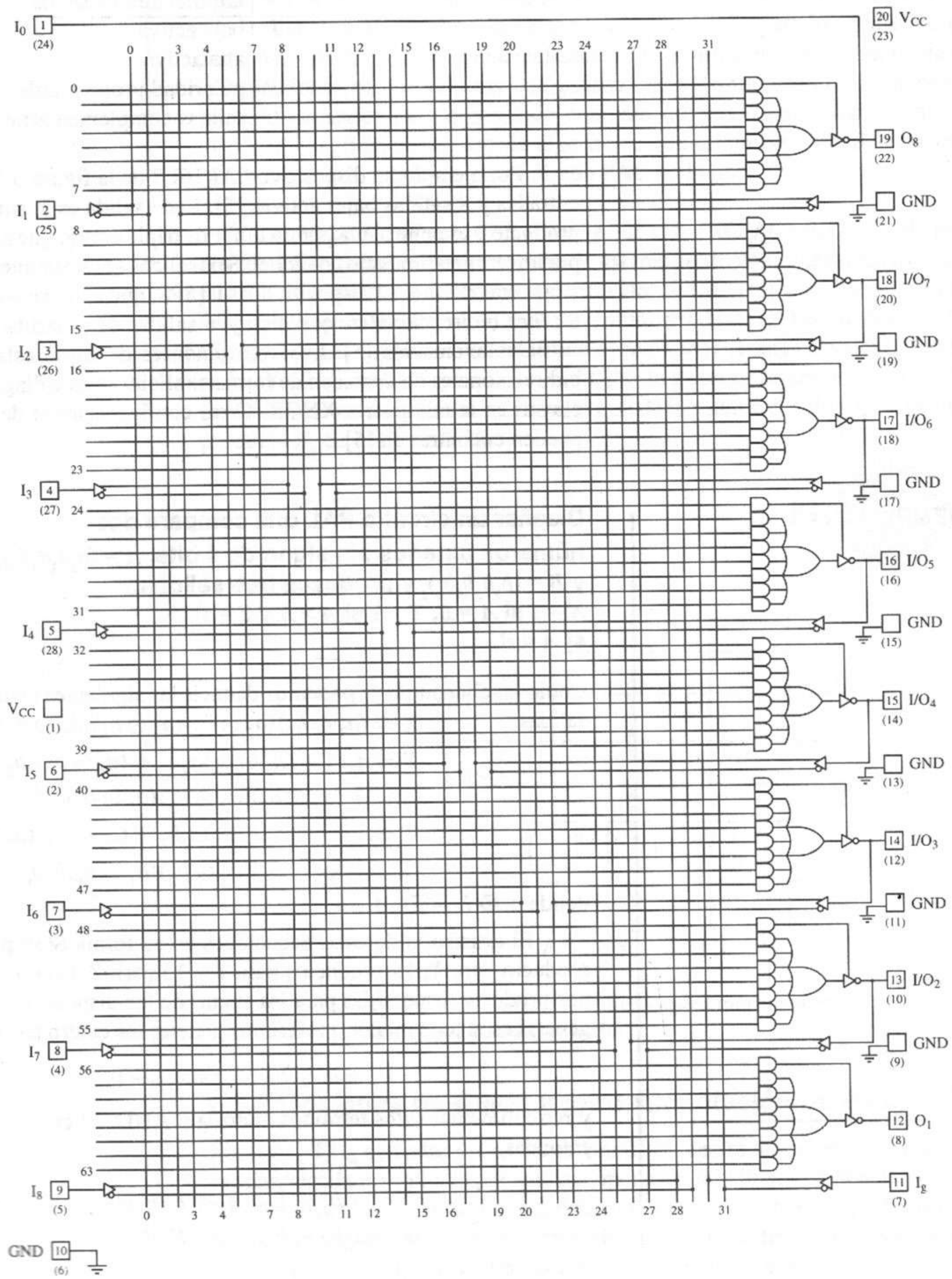


Figura 5.32 PAL16L8: salidas bajas activas [5]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

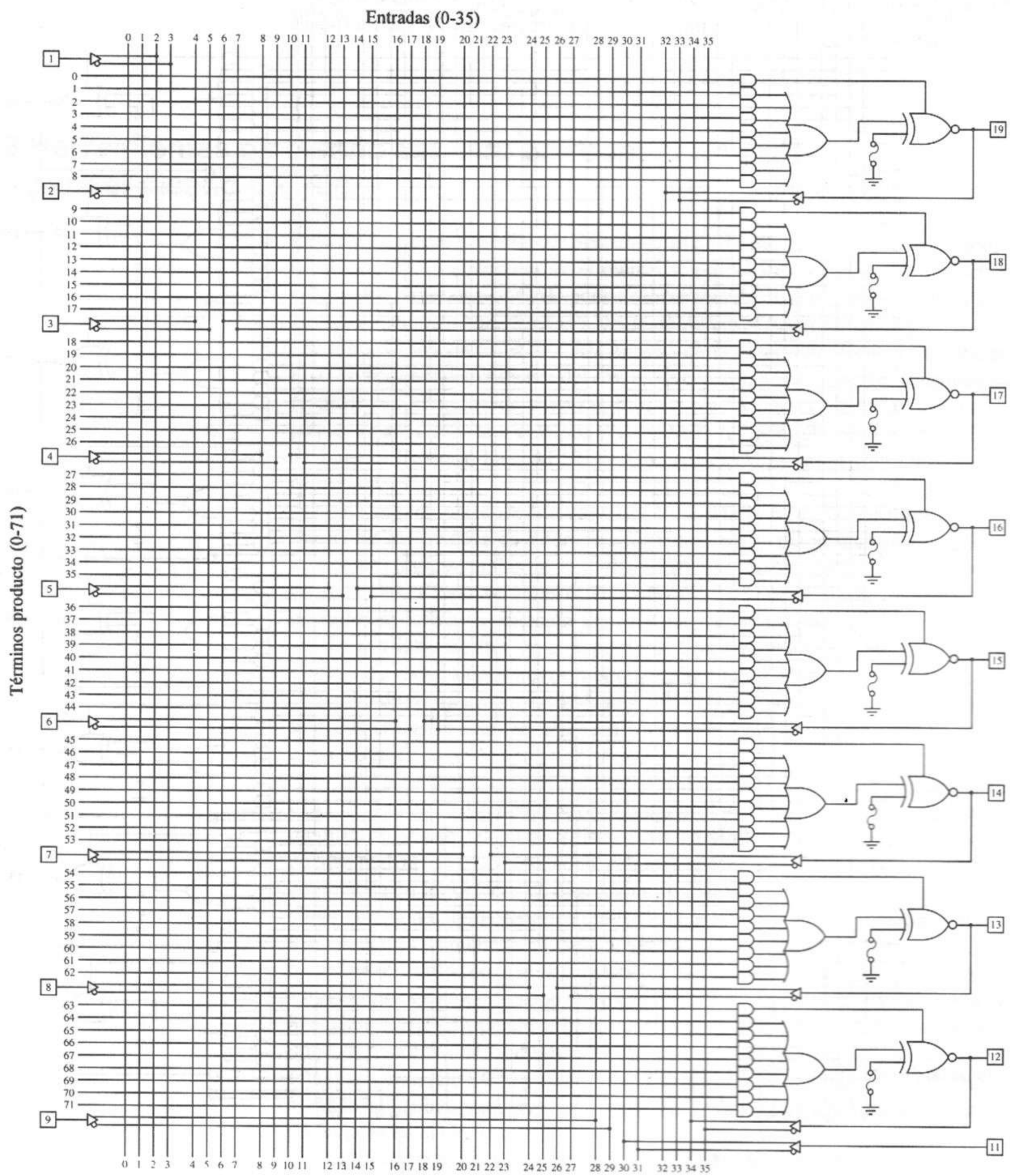


Figura 5.33 PAL18L8: salidas con polaridad programable [5]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

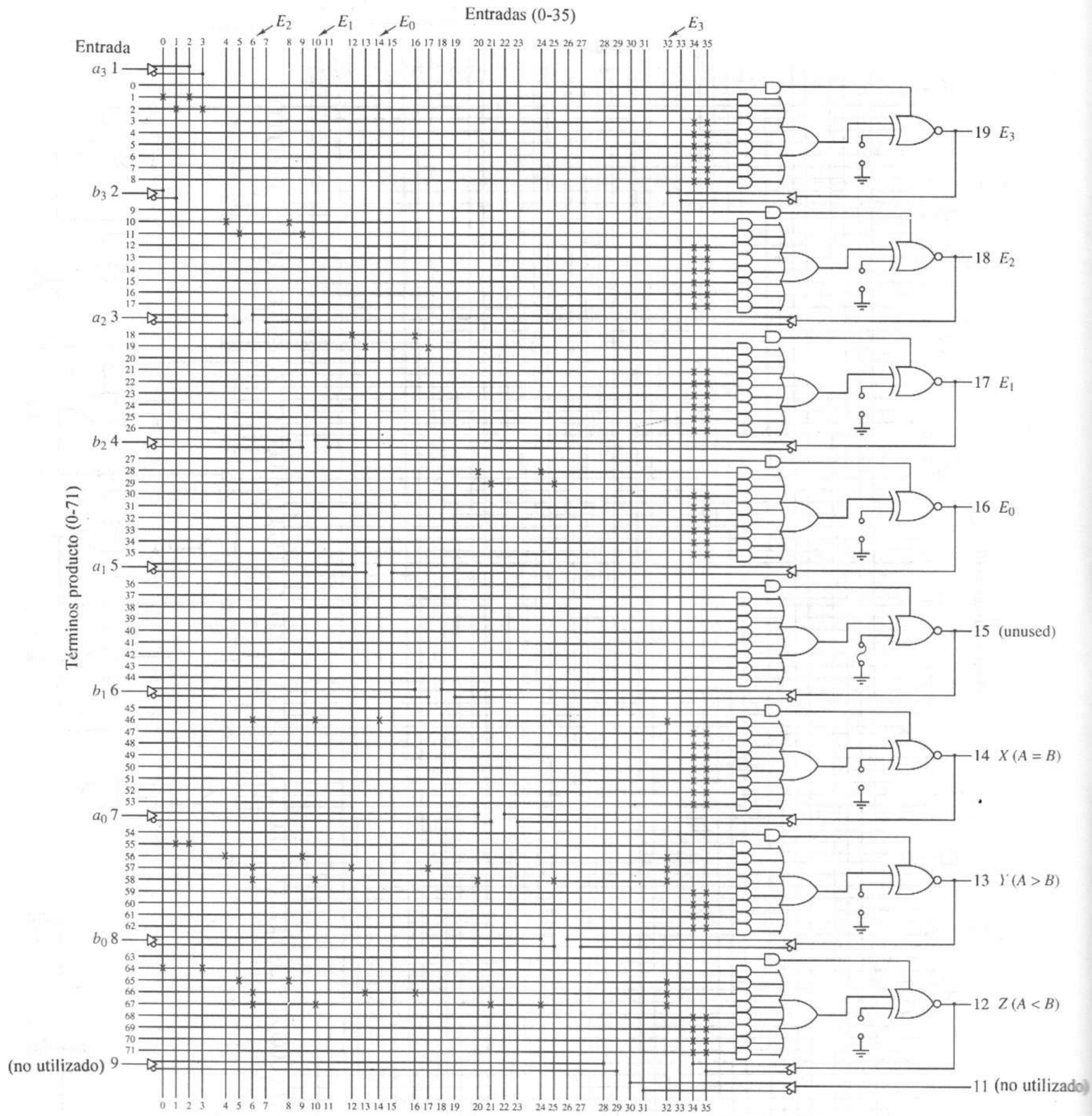


Figura 5.34 Comparador de 4 bits asociado a un dispositivo PAL18P8 [5]. Copyright © Advanced Micro Devices, Inc., 1993 (adaptado). Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

Otras opciones de salida PAL incluyen elementos de memoria, llamados *flip-flops* y *latches*, para apoyar el diseño de circuitos secuenciales. En el capítulo 6 presentaremos los *flip-flops* y los *latches*, y en el capítulo 11 analizaremos los dispositivos PAL que incluyen estos elementos.

5.6 Herramientas de diseño asistido por computadora para el diseño de PLD

Existen dos clases generales de herramientas de diseño asistido por computadora (CAD) de PLD disponibles en el mercado: específicas y universales. Muchos ofrecen herramientas de CAD para el desarrollo de diseños sólo para sus PLD. Algunos ejemplos son MAX+PLUS II de Altera y AMAZE de Signetics. Sin embargo, varios sistemas de CAD apoyan el desarrollo de diseños basados en PLD en forma independiente del dispositivo y después asocian los diseños con dispositivos elegidos de diversas bibliotecas. Algunos ejemplos típicos de paquetes para diseño universal son PALASM de Advanced Micro Devices, CUPL de Logical Devices, Inc., ABLE de DATA I/O Corporation y PLDesigner de Minc, Inc.

La mayor parte de los paquetes de CAD para PLD permiten crear e introducir los diseños en diferentes formatos, como diagramas esquemáticos, ecuaciones lógicas, tablas de verdad y diagramas y tablas de estado de circuitos secuenciales. Como se muestra en la figura 5.35, cada diseño se traduce, o *compila*, en forma de ecuación lógica y después se minimiza usando métodos similares a los descritos en el capítulo 3. Con frecuencia, el diseñador puede elegir entre varios algoritmos de minimización y debe decidir si es más importante el tiempo de cómputo o la optimización de los resultados. El diseño compilado se puede entonces simular para verificar que sea correcto y evaluar el tiempo y otros parámetros.

Si el diseño es correcto, las ecuaciones lógicas se asocian con el dispositivo PLD elegido. Si el diseño no puede ajustarse al PLD elegido, el diseñador debe modificar el diseño, elegir otro dispositivo o dividir el diseño en módulos que se puedan realizar en PLD separados. Algunos sistemas de CAD buscan de manera automática en bibliotecas de dispositivos e identifican los PLD más adecuados y que cumplen con los criterios específicos. Algunos de estos sistemas pueden incluso dividir de manera automática un diseño para asociarlo con diversos PLD o combinar diseños pequeños para ajustarlos a un único PLD. El resultado de la etapa de adecuación del dispositivo es un *mapa de fusibles*, que es un mapa de los fusibles del PLD, donde se indica cuáles serán eliminados y cuáles se dejarán intactos para realizar el diseño. En la mayor parte de los casos, se utiliza un estándar, como JEDEC [5], para el mapa de fusibles. A continuación se descarga el mapa de fusibles a un programador de PLD especial para programar el patrón de fusibles en el circuito.

La mayor parte de los paquetes de diseño de PLD utilizan un lenguaje de alto nivel para expresar los diseños en formato de ecuación lógica, tabla de verdad o máquina de estado de circuito secuencial. Muchos de ellos también aceptan diseños creados con programas de captura de esquemas. En estos casos, el esquema se traduce a una forma de ecuación lógica en el lenguaje utilizado por ese paquete.

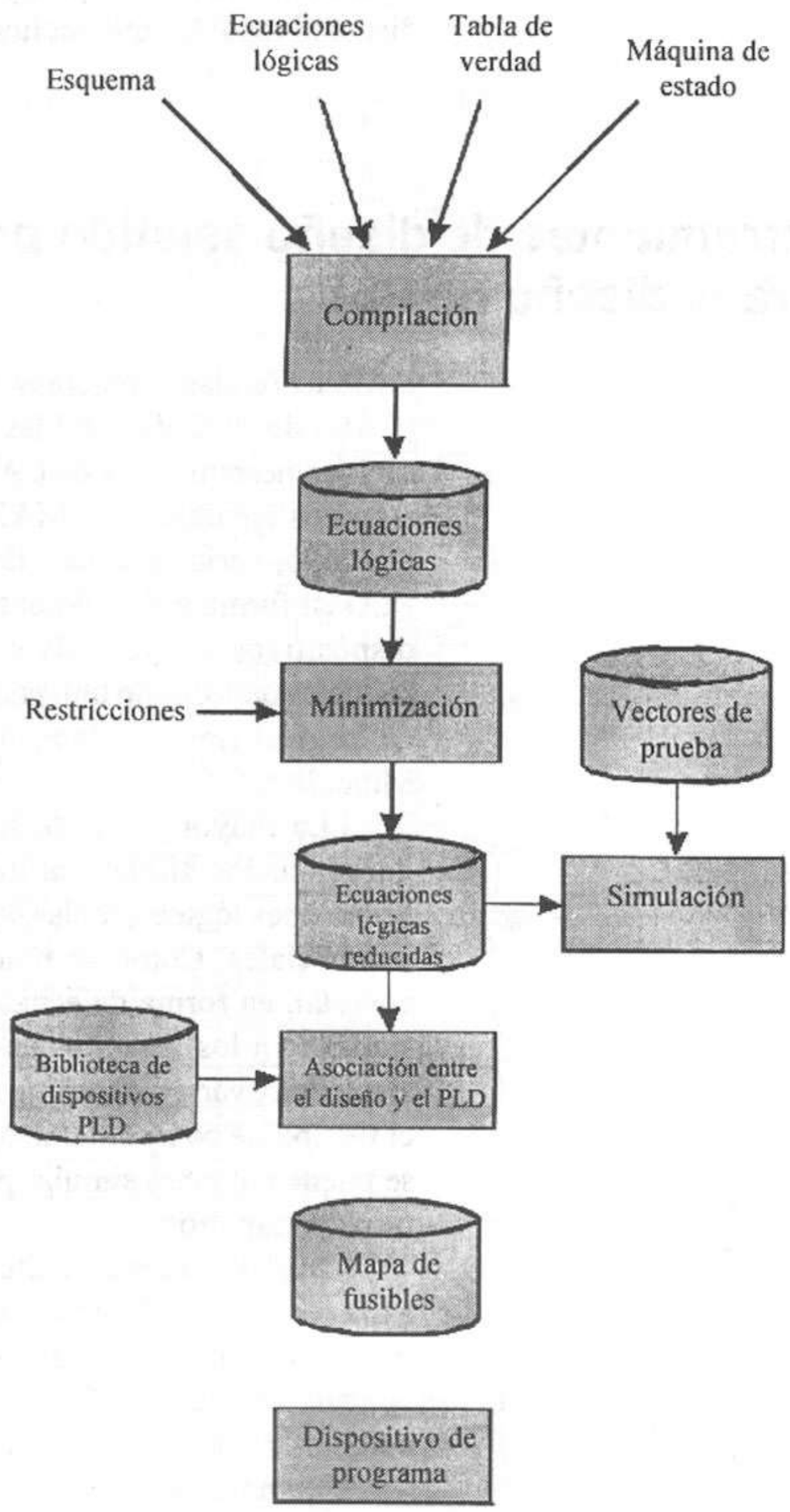


Figura 5.35 Proceso de diseño de PLD.

Por ejemplo, la figura 5.36 muestra un diagrama esquemático para un sumador completo de 1 bit. Este diagrama fue creado con el editor de esquemas *Design Architect* de MentorGraphics y después traducido por el programa *PLDesigner* de Minc al lenguaje PDL. La figura 5.37 muestra el listado de PLD resultante y la figura 5.38 muestra las ecuaciones lógicas reducidas producidas por el compilador de PDL.

En la siguiente sección presentamos un panorama del lenguaje PDL. Otros lenguajes de diseño de PLD son similares a PDL. El lector puede consultar [6,7,8] si desea más detalles.

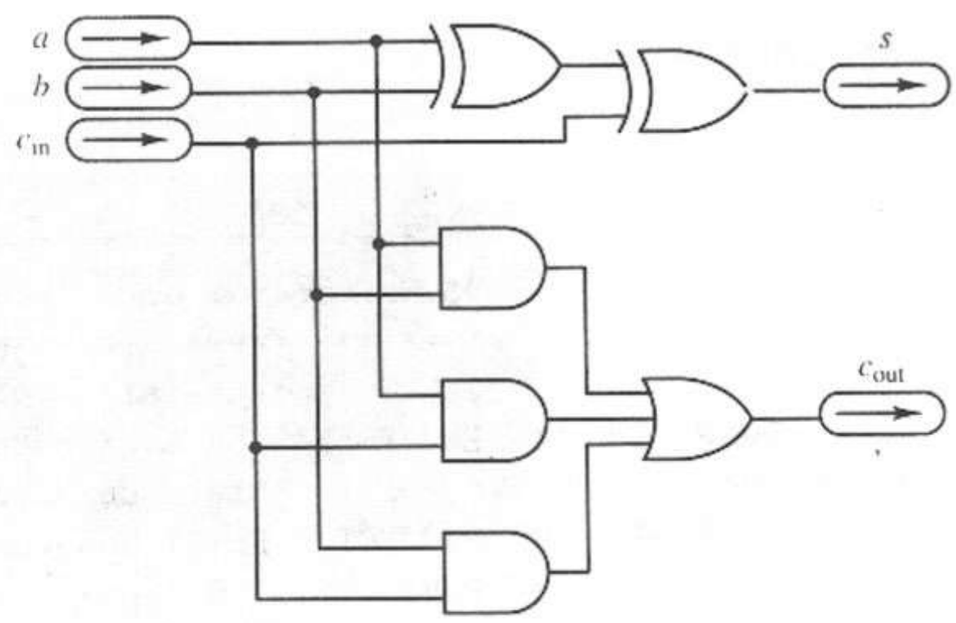


Figura 5.36 Diagrama esquemático de un sumador completo de 1 bit.

5.6.1 Representación de diseño con PDL

PDL (PLDesigner Design Language) es representativo de los lenguajes de alto nivel utilizados por la mayor parte de las herramientas de diseño de PLD. Podemos introducir los diseños en forma de ecuación, de tabla de verdad, de diagrama de estado, de tabla de estado y otras formas de comportamiento. Como muestra el ejemplo de la figura 5.37, un archivo en PDL tiene una sección de encabezado, con una descripción verbal del diseño, una sección opcional de definición de macros que permite la representación simbólica de funciones y expresiones, y una sección de definición de funciones con declaraciones de señales de entrada y salida y las ecuaciones lógicas, tablas de verdad o descripciones de la máquina de estado que describen las funciones por realizar. Observe que en todo el archivo de diseño se pueden incluir comentarios, cada uno de los cuales comienza con comillas.

Declaraciones de señales de entrada y salida

Todo diseño tiene varias entradas y salidas externas y, en algunos casos, líneas de entrada/salida bidireccionales. En un archivo PDL estas señales se definen, o *declaran*, antes de indicar la descripción funcional del diseño. En los dispositivos lógicos programables las entradas externas pueden ser *pines* exclusivos de entrada o líneas de entrada/salida cuyos controladores de salida están desactivados. Los siguientes ejemplos ilustran algunos formatos de declaración de señales de entrada y salida.

```
INPUT x,y,[I3..I0];            "entradas exclusivas:
OUTPUT x,[c3..c0];             "salidas combinatorias
OUTPUT x,y ENABLED_BY oe;      "salidas combinatorias con
                                controladores de triple estado
BIPUT x1,x2 ENABLED_BY oe;     "línea de entrada/salida
```

En estos ejemplos, observe que podemos definir las señales numeradas en forma secuencial mediante notación de intervalo. Por ejemplo, [*c*3..*c*0] representa las cuatro señales *c*3, *c*2, *c*1 y *c*0. La palabra reservada ENABLED_BY indica un

```
"=========================================
"Sección de encabezado
"=========================================
TITLE      schematic.vpt ;
ENGINEER   Joe E. Student;
COMPANY    State University ;
PROJECT    EE401 Homework Project ;
REVISION   1.0 ;
COMMENT Circuito sumador completo de 1 bit;

"=========================================
"Sección de definición de macros
"=========================================
"Macros para las compuertas AND, OR y XOR

MACRO AND2(i0,i1) (i0 * i1) ;
MACRO OR3(i0,i1,i2) (i0 + i1 + i2) ;
MACRO XOR(i0,i1)  (i0 (+) i1) ;

"=========================================
"Sección de definición de funciones
"=========================================
FUNCTION schematic ;

"Declaración de señales de entrada y salida externas
   INPUT   A,B,CIN ;
   OUTPUT  COUT,S ;

"Instancias de tres compuertas AND
   MACRO  N$11  AND2(A,B) ;
   MACRO  N$12  AND2(A,CIN) ;
   MACRO  N$13  AND2(B,CIN) ;

"Instancia de una compuerta OR
   MACRO  COUT  OR3(N$11,N$12,N$13) ;

"Instancias de dos compuertas XOR
   MACRO  N$14  XOR(A,B) ;
   MACRO  S     XOR(N$14,CIN) ;

END schematic ;
```

Figura 5.37 Descripción en PDL de un sumador completo de 1 bit generado a partir de su esquema.

controlador de triple estado asociado a una salida y define la señal de control para el controlador. La palabra reservada ENABLED_BY se puede utilizar u omitir según sea necesario para ajustarse a las salidas reales de un dispositivo lógico en particular.

```
S.EQN        = CIN*/B*/A
             + /CIN*B*/A
             + /CIN*/B*A
             + CIN*B*A ; "(4 términos)

COUT.EQN     = A*CIN
             + B*CIN
             + B*A ; "(3 términos)
```

Figura 5.38 Ecuaciones en PDL para el sumador completo de 1 bit generado por el compilador de PDL.

TABLA 5.5 OPERADORES LÓGICOS DEL LENGUAJE PLD

Símbolo	Operación lógica	Ejemplo
/	NOT	/a
/*	NAND	a /* b
/+	NOR	a /+ b
*	AND	a * b
+	OR	a /+ b
(+)	XOR	a (+) b
/(+)	XNOR	a /(+) b
[+]	XOR por hardware	a [+] b

Ecuaciones lógicas

Las ecuaciones lógicas se expresan en PDL exactamente como si se escribieran en papel. La tabla 5.5 enumera los operadores lógicos disponibles en PDL, en orden de precedencia descendente. También se pueden utilizar paréntesis en caso necesario. La figura 5.39 muestra las ecuaciones lógicas de un circuito sumador completo de 1 bit, tal como se introducirían en PDL.

Las ecuaciones en PDL se pueden expresar en cualquier formato, desde expresiones SOP o POS sencillas hasta expresiones complejas de varios niveles. Al compilar un diseño, todas las ecuaciones se convierten a la forma SOP de dos niveles necesaria para adecuarse a los arreglos AND/OR de los dispositivos

```
s      = (a(+)b)(+)cin;
cout   = (a*b) + (a*cin) + (b*cin);
```

(a)

```
S.EQN        = CIN*/B*/A
             + /CIN*B*/A
             + /CIN*/B*A
             + CIN*B*A ;
COUT.EQN     = A*CIN
             + B*CIN
             + B*A ;
```

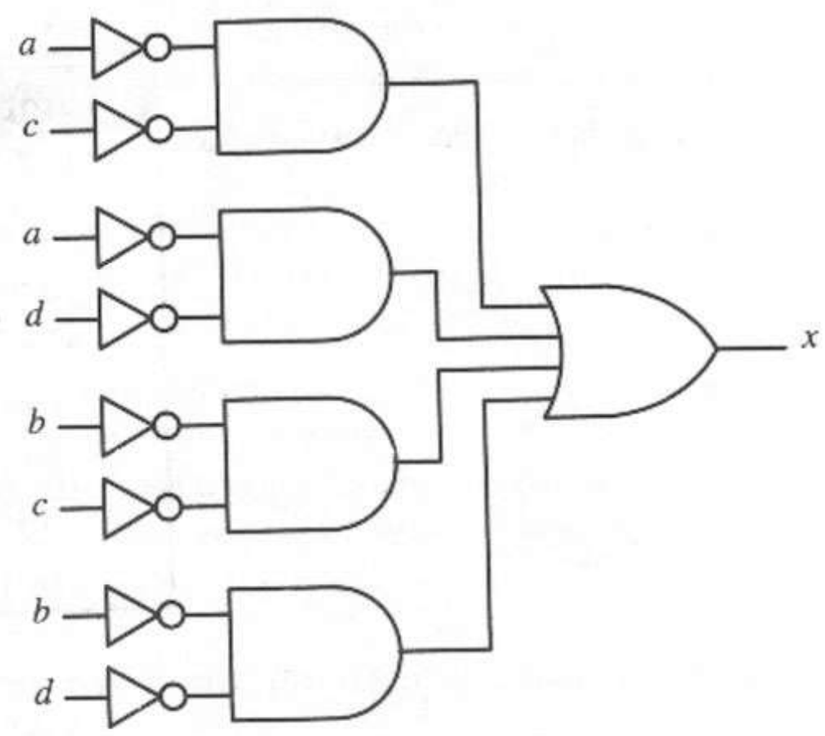
(b)

Figura 5.39 Sumador completo representado en PDL con ecuaciones lógicas. **(a)** Ecuaciones lógicas PDL. **(b)** Ecuaciones producidas por el compilador de PDL.

PAL y PLA. Por ejemplo, la figura 5.39b muestra la salida del compilador de PDL para las ecuaciones del sumador completo de la figura 5.39a.

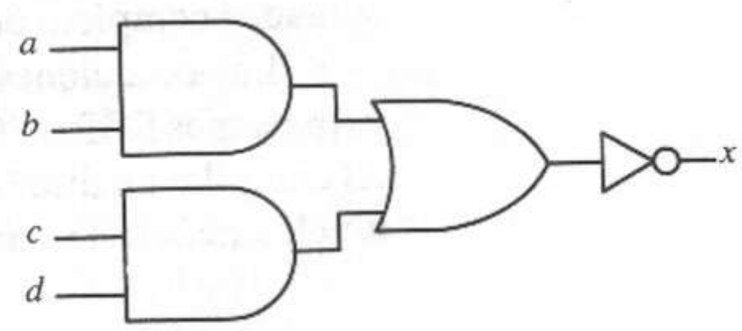
Como ayuda para el desarrollo de las ecuaciones lógicas y su asociación a un dispositivo en particular, cualquier línea de entrada, salida o entrada/salida bidireccional se puede definir como baja activa. En el caso de las salidas de dispositivos lógicos con controladores de inversión, muchas veces conviene

```
OUTPUT x;
x = /(a*b + c*d);
```
(a)



(b)

```
LOW_TRUE OUTPUT x;
x = a*b + c*d;
```
(c)



(d)

```
OUTPUT x;
/x = a*b + c*d;
```
(e)

Figura 5.40 Uso de salidas de dispositivos altas activas y bajas activas. **(a)** Descripción en PDL de $x = \overline{ab + cd}$. **(b)** Diagrama esquemático (salida alta activa). **(c)** Descripción en PDL de $\overline{x} = ab + cd$. **(d)** Diagrama esquemático (salida invertida). **(e)** Forma PDL alternativa de $\overline{x} = ab + cd$.

definir la salida como baja activa. Por ejemplo, la figura 5.40 muestra varias representaciones de la misma expresión de conmutación.

$$x = \overline{ab + cd} = (\bar{a} + \bar{b})(\bar{c} + \bar{d})$$

La forma de la figura 5.40a indicaría al compilador que transforme la expresión a su forma SOP mediante el teorema de DeMorgan, lo que produce el circuito AND/OR de la figura 5.40b. Si el dispositivo lógico objetivo tiene un controlador de inversión sobre la salida, la forma de la figura 5.40c indica al compilador que utilice el arreglo AND/OR para formar la expresión SOP $ab + cd$ y suponga que un controlador de salida lo invertirá, como en la figura 5.40d. También se puede designar una señal como baja activa dentro de una proposición de asignación de señal, como en la figura 5.40e, la cual produce el mismo resultado que el listado de la figura 5.40c.

Macros

Una *macro* es un mecanismo para representar de manera simbólica funciones que serán utilizadas varias veces (como las expresiones de conmutación realizadas por diversas compuertas lógicas) y para asignar símbolos que remplacen diversos patrones y mejoren la legibilidad de una descripción en PDL. Cada instancia de una macro se remplaza por su definición durante la compilación de un diseño, y todos los parametros formales se remplazan con sus valores reales. El formato de una definición de macro es el siguiente:

MACRO nombre-de-macro [(parámetros)]texto;

El listado de la figura 5.37 contiene tres definiciones de macro, cada una de las cuales describe uno de los elementos de circuito en el diagrama esquemático de la figura 5.36. Por ejemplo, la compuerta AND de dos entradas se define como

```
MACRO AND2(i0,i1) (i0 * i1 ) ;
```

lo que define la función AND2 como el AND de dos parámetros $i0$ e $i1$. Utilizamos tres instancias de AND2 en la descripción funcional del sumador completo.

```
MACRO  N$11  AND2(A,B) ;
MACRO  N$12  AND2(A,CIN) ;
MACRO  N$13  AND2(B,CIN) ;
```

Esto define tres compuertas AND de dos entradas cuyas expresiones lógicas equivalentes son

```
N$11 = A * B ;
N$12 = A * CIN ;
N$13 = B * CIN ;
```

De manera análoga, podemos describir las compuertas OR y XOR utilizadas en la figura 5.36 mediante definiciones de macro, cuyas instancias utilizamos en la sección de definición de funciones para crear las compuertas lógicas *COUT*, *N*$14 y *S*, cuyas entradas son controladas por las entradas del circuito y por las salidas de las compuertas AND *N*$11, *N*$12 y *N*$13. La figura 5.38 muestra esto y las ecuaciones desarrolladas generadas a partir del listado en PDL de la figura 5.37.

Tablas de verdad

Como se explicó en el capítulo 2, una tabla de verdad enumera todas las combinaciones de las variables de entrada de una función lógica y el valor de la función para cada combinación. Por economía, con frecuencia se enumeran varias funciones en la misma tabla de verdad. Por ejemplo, la figura 5.41a muestra la tabla de verdad del sumador completo de la figura 5.39a.

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a)

```
TRUTH_TABLE
  a,  b,  c_in  ::  c_out,  s;
  0,  0,  0     ::  0,     0;
  0,  0,  1     ::  0,     1;
  0,  1,  0     ::  0,     1;
  0,  1,  1     ::  1,     0;
  1,  0,  0     ::  0,     1;
  1,  0,  1     ::  1,     0;
  1,  1,  0     ::  1,     0;
  1,  1,  1     ::  1,     1;
END;
```

(b)

Figura 5.41 Tabla de verdad del sumador completo. **(a)** Tabla de verdad. **(b)** Formato de tabla de verdad en PDL.

La figura 5.41b muestra la descripción en PDL de la tabla de verdad del sumador completo. La primera línea define las variables de entrada y salida, separadas por dos símbolos de dos puntos (::). En cada una de las líneas que siguen se enumera una combinación de variables de entrada, seguida por dos signos de dos puntos y después los valores de salida correspondientes.

En caso necesario, se indica una condición prescindible en la tabla de verdad con × y un valor de impedancia alta con Z. Los valores de salida también se pueden remplazar por expresiones lógicas de las variables de entrada.

5.6.2 Procesamiento de un archivo de diseño PDL

Después de preparar la descripción de un diseño en PDL, el compilador de PDL debe traducir y reducir el diseño. Esto implica varios pasos. En el caso de una descripción de comportamiento, como una tabla de verdad, una tabla de estado

u otra descripción de máquina de estado, el compilador primero sintetiza la máquina convirtiendo dicha descripción en ecuaciones lógicas para todas las salidas y las entradas de excitación de *flip-flops*. Una vez que el diseño tiene la forma de ecuación lógica, las ecuaciones se simplifican a formas SOP de dos niveles, las cuales se pueden asociar al arreglo AND/OR de un dispositivo PAL o PLA elegido.

En el proceso de simplificación de las ecuaciones, éstas se minimizan mediante una de cuatro opciones que puede especificar el usuario. La primera no realiza reducción alguna, sino que deja las ecuaciones en forma SOP. La segunda opción aplica el algoritmo *ESPRESSO*, que reduce las ecuaciones con rapidez y poco uso de memoria, pero que no necesariamente produce una solución óptima. La tercera opción utiliza el algoritmo *ESPRESSO* aplicando algunas técnicas de *Quine-McCluskey* para obtener una mejor cubierta. La cuarta opción utiliza el método completo de *Quine-McCluskey*, que proporciona una solución óptima pero a costa de un mayor tiempo de cálculo y mayor uso de memoria.

Por ejemplo, el compilador de PDL genera las ecuaciones lógicas de la figura 5.42 a partir de la tabla de verdad de la figura 5.41 mediante el algoritmo de minimización ESPRESSO.

```
COUT.EQN   = A*CIN
           + B*CIN
           + B*A ; "(3 términos)

S.EQN      = CIN*/B*/A
           + /CIN*B*/A
           + /CIN*/B*A
           + CIN*B*A ; "(4 términos)
```

Figura 5.42 Ecuaciones de excitación reducida y de salida para el sumador completo.

Después de compilar el diseño, en el siguiente paso verificamos que sea correcto mediante la simulación funcional. El lenguaje PDL permite especificar vectores de prueba y controles de simulación dentro del archivo de diseño, de modo que la simulación puede ocurrir en forma inmediata después de la compilación, con las herramientas *PLDsim* del sistema PLDsynthesis. Cuando PLDsynthesis se integra en otro ambiente de diseño, como el simulador lógico *Falcon Framework* de Mentor Graphics, se pueden utilizar otros simuladores, como el *QuickSim II* de Mentor Graphics. El lector puede consultar [7] si desea más detalles acerca de la simulación dentro de los ambientes PLDesigner y Falcon Framework.

El siguiente paso del proceso consiste en asociar las ecuaciones reducidas al dispositivo elegido. El sistema PLDesigner incluye una biblioteca de dispositivos de donde se pueden seleccionar los dispositivos más adecuados para un diseño y que cumplan criterios especificados por el usuario.

En PLDsynthesis, estos criterios especificados por el usuario, o valores de restricción, incluyen el tipo de paquete, la familia lógica, el fabricante, la especificación de temperatura, el máximo de corriente, la frecuencia máxima, el retardo máximo y el precio del componente. A cada restricción se le asigna un factor de ponderación, de modo que en la selección de un dispositivo se otorgue más importancia a los valores de restricción que el diseñador considere críticos.

La salida de la operación de ajuste al dispositivo es un mapa de fusibles, que puede introducirse en un programador de dispositivos para que programe el circuito. En ciertos casos, el programador de dispositivos puede recibir la información de simulación para que pueda verificar el dispositivo y comparar el funcionamiento real con los resultados simulados.

5.7 Resumen

En este capítulo examinamos el uso de dispositivos lógicos programables para la implantación de circuitos lógicos combinatorios. Presentamos las estructuras básicas de circuito de los tres tipos de PLD, y describimos el proceso de asociación de funciones lógicas a cada uno, junto con varios ejemplos. Por último, describimos las herramientas de diseño asistido por computadora utilizadas para el desarrollo de circuitos PLD. En el capítulo 11 analizaremos otros dispositivos programables que apoyan el diseño de circuitos tanto secuenciales como combinatorios.

REFERENCIAS BIBLIOGRÁFICAS

1. PAUL M. CHIRLIAN, *Analysis and Design of Integrated Electronic Circuits*, 2nd ed. New York: Harper & Row, 1987.
2. ROGER C. ALFORD, *Programmable Logic Designer's Guide*. Indianapolis, IN: Howard W. Sams, 1989.
3. PARAG K. LALA, *Digital System Design Using Programmable Logic Devices*. Englewood Cliffs, NJ: Prentice Hall, 1990.
4. PHILIPS, *Programmable Logic Devices (PLD) Data Handbook*. Sunnyville, CA: Philips Semiconductor, 1994.
5. MONOLITHIC MEMORIES, *PAL Programmable Array Logic Handbook*, 2nd ed. Monolithic Memories, Inc., 1981.
6. MINC, INC., *PLDesigner Student Version Manual*. New York: McGraw-Hill Publishing Co., 1990.
7. *PLDsynthesis User's Manual*, Wilsonville, OR: Mentor Graphics Corp.
8. *PDL Language Reference Manual*, Wilsonville, OR: Mentor Graphics Corp.

PROBLEMAS

5.1 Diseñe un convertidor de código BCD a exceso 3 mediante:

(a) Una red lógica (compuertas NAND de dos niveles)

(b) PLA (como en la figura 5.7)

(c) ROM (como en la figura 5.25)

(d) PAL (como en la figura 5.30)

Recuerde especificar las dimensiones de la entrada y la salida de su PLA, ROM y PAL.

5.2 Implante las siguientes funciones mediante:

(a) Un decodificador 4 a 16 y compuertas lógicas

(b) PLA (como en la figura 5.7)

(c) ROM (como en la figura 5.25)

(d) PAL (como en la figura 5.30)

$$f_1(A, B, C, D) = \sum m(0, 1, 2, 3, 6, 9, 11)$$
$$f_2(A, B, C, D) = \sum m(0, 1, 6, 8, 9)$$
$$f_3(A, B, C, D) = \sum m(2, 3, 8, 9, 11)$$

5.3 Utilice una ROM de 32×6 para convertir un número binario de 6 bits en su representación BCD correspondiente de 2 dígitos.

$$(a_5a_4a_3a_2a_1a_0)_2 = [(x_3x_2x_1x_0)_{BCD}(y_3y_2y_1y_0)_{BCD}]_{10}$$

Muestre el contenido de la ROM en el formato de tabla de verdad. (*Sugerencia*: $x_3 = 0$ y $y_0 = a_0$.)

5.4 Muestre la implantación de las funciones del problema 5.2 mediante:

(a) FPLA PLS100

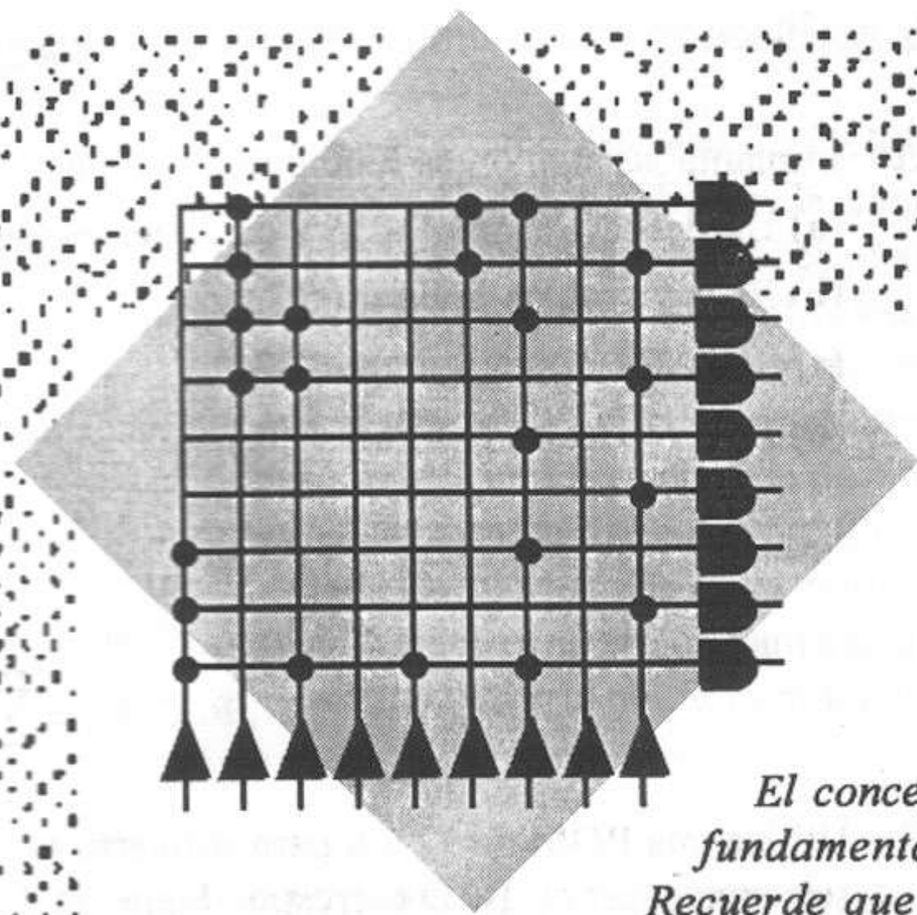
(b) PAL16L8

5.5 Muestre la forma de implantar un restador binario mediante:

(a) FPLA PLS100

(b) PAL16L8

6



El concepto básico de dispositivo secuencial es fundamental en el diseño de sistemas digitales. Recuerde que en las redes lógicas combinatorias la salida sólo es función de la entrada presente. En contraste, la salida de un dispositivo secuencial no sólo depende de la entrada presente, sino también de las entradas anteriores. Este historial de entradas de un circuito secuencial se conserva mediante el uso de dispositivos de almacenamiento, es decir, de memoria. *Este uso de memoria añade una nueva dimensión al diseño lógico, proporcionando la capacidad de resolver diversos problemas para cuya solución no basta la lógica combinatoria.*

En este capítulo presentaremos el modelo de circuito secuencial básico, y después describiremos el diseño y funcionamiento de varios elementos de memoria comunes, como los latches y flip-flops.

Introducción a los dispositivos secuenciales

6.1 Modelos para circuitos secuenciales

El concepto secuencial no se restringe a los sistemas digitales. Por ejemplo, consideremos el funcionamiento de un elevador en un edificio de cuatro pisos. El elevador actúa como un dispositivo secuencial, ya que sus acciones están determinadas por señales de entrada de sus tableros de control (en el elevador y en cada piso) y por su posición actual, en el piso 1, 2, 3 o 4. El elevador debe "recordar" su posición actual para determinar su siguiente transición de piso. Por tanto, definimos el *estado presente* del elevador como una descripción de su posición de piso actual, con un historial de sus transiciones de piso pasadas. Por ejemplo, el elevador podría estar "en el piso 3 y subiendo". Este estado presente se debe diferenciar de "en el piso 3 y bajando". El *siguiente estado* (y, por tanto, la siguiente posición de piso) del elevador está determinado por su estado actual y su *entrada*, la cual consiste en la situación de los botones de los tableros de control localizados en el elevador y en cada piso. Si el elevador está "en el piso 3 y bajando", responderá a una solicitud de bajar hecha en el piso 2, ¡pero ignorará una solicitud de subir hecha en el piso 2! Una vez determinado el siguiente estado, se ordena un *estado de transición* mediante una orden al motor, el cual lleva el elevador a un nuevo piso. Los conceptos de estado presente, siguiente estado, entrada y estado de transición son fundamentales en el estudio de los circuitos lógicos secuenciales.

Otro ejemplo sencillo de dispositivo secuencial, y que tiene una amplia gama de aplicaciones en los sistemas digitales, es un contador. Podemos utilizar este dispositivo para realizar funciones como totalizar el número de carros que entran en un estacionamiento o llevar un registro de ciertas funciones que deben realizarse dentro de un sistema de computación de gran tamaño. Estudiaremos con detalle los contadores en el siguiente capítulo.

6.1.1 Representación de diagrama de bloques

En nuestro estudio de las redes lógicas combinatorias, vimos que podemos representar estos circuitos como en la figura 6.1a. La relación matemática que describe esta red es

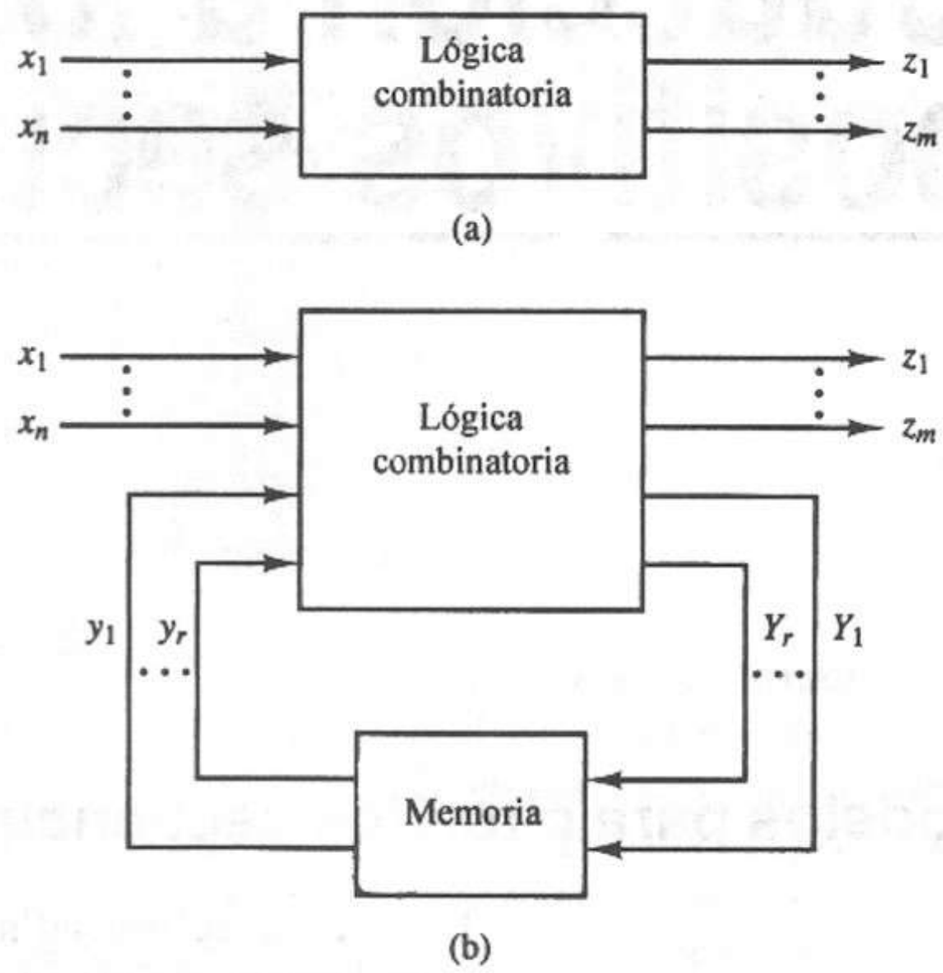


Figura 6.1 Modelo de circuito secuencial. **(a)** Circuito lógico combinatorio. **(b)** Circuito lógico secuencial.

$$z_i = f_i(x_1, x_2, \ldots, x_n), \qquad i = 1, \ldots, m \tag{6.1}$$

Esta ecuación establece que la salida es función únicamente de la entrada actual. Todas las señales de la ecuación 6.1 asumen uno de los dos valores 0 o 1.

La figura 6.1b muestra el modelo de un circuito secuencial. Las n-adas $(x_1,\ldots,x_n)$ son la *entrada*, las m-adas $(z_1,\ldots,z_m)$ son la *salida*, y las r-adas $(y_1,\ldots,y_r)$ y $(Y_1,\ldots,Y_r)$ representan el *estado presente* y el *siguiente estado*, respectivamente. Podemos expresar las relaciones entre estas variables en forma matemática como

$$z_i = g_i(x_1, \ldots, x_n, y_1, \ldots, y_r), \qquad i = 1, \ldots, m \tag{6.2}$$

$$Y_i = h_i(x_1, \ldots, x_n, y_1, \ldots, y_r), \qquad i = 1, \ldots, r \tag{6.3}$$

donde g_i y h_i son funciones booleanas. Podemos escribir las ecuaciones 6.2 y 6.3 en notación vectorial como

$$\mathbf{z} = \mathbf{g}(\mathbf{x}, \mathbf{y}) \tag{6.4}$$

$$\mathbf{Y} = \mathbf{h}(\mathbf{x}, \mathbf{y}) \tag{6.5}$$

donde

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix}, \qquad \mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_r \end{bmatrix} \tag{6.6}$$

Observe que z_i, x_i, y_i y Y_i son variables binarias (sus valores son 0 lógico o 1 lógico).

Todos los vectores de la ecuación 6.6 dependen del tiempo; convendremos que el vector **y** tiene el valor $\mathbf{y}(t_k)$ en el instante t_k. Ocasionalmente, examinaremos una señal $\mathbf{y}(t)$ a intervalos de tiempo con una longitud fija. Si $t_k = k\Delta t$ (k entero), entonces

$$\mathbf{y}(t_k) = \mathbf{y}(k\Delta t) \triangleq \mathbf{y}^k \tag{6.7}$$

donde Δt es un incremento de tiempo fijo.

Los dispositivos de memoria en el diagrama de bloques de la figura 6.1b pueden ser de varios tipos: *flip-flops* semiconductores, dispositivos magnéticos, líneas de retardo, relevadores mecánicos, conmutadores de rotación, y muchos otros. Analizaremos más adelante muchos de los dispositivos de memoria fabricados con semiconductores.

Las señales de entrada x_i y las señales de salida z_j de la figura 6.1 también pueden asumir varias formas, que analizaremos posteriormente.

6.1.2 Tablas y diagramas de estados

Las ecuaciones lógicas 6.2 y 6.3 y las ecuaciones vectoriales 6.4 y 6.5 definen por completo el comportamiento del circuito secuencial modelado en la figura 6.1b para un dispositivo de memoria dado. Sin embargo, la descripción, si bien es completa, no presenta una imagen clara de las relaciones entre las variables pertinentes. La relación funcional existente entre la entrada, la salida, el estado presente y el siguiente estado se muestran de manera clara en la tabla de estados o en el diagrama de estados. El *diagrama de estados* es una representación gráfica de un circuito secuencial, en la que los estados del circuito aparecen como círculos y las transiciones de estado (el paso del estado presente **y** al siguiente estado **Y**) se indican con flechas. Cada flecha está rotulada con la entrada **x** y la salida del circuito correspondiente **z**, como en la figura 6.2a.

La figura 6.2b muestra la representación mediante una *tabla de estados*. Todos los vectores de entrada del circuito **x** se enumeran en la parte superior, mientras que todos los vectores de estado **y** se enumeran del lado izquierdo. Las entradas de la tabla son el siguiente estado **Y** y la salida **z**. Leemos la tabla como sigue: Para una entrada **x** con un circuito secuencial en el estado **y**, el circuito pasará al siguiente estado **Y** con una salida **z**.

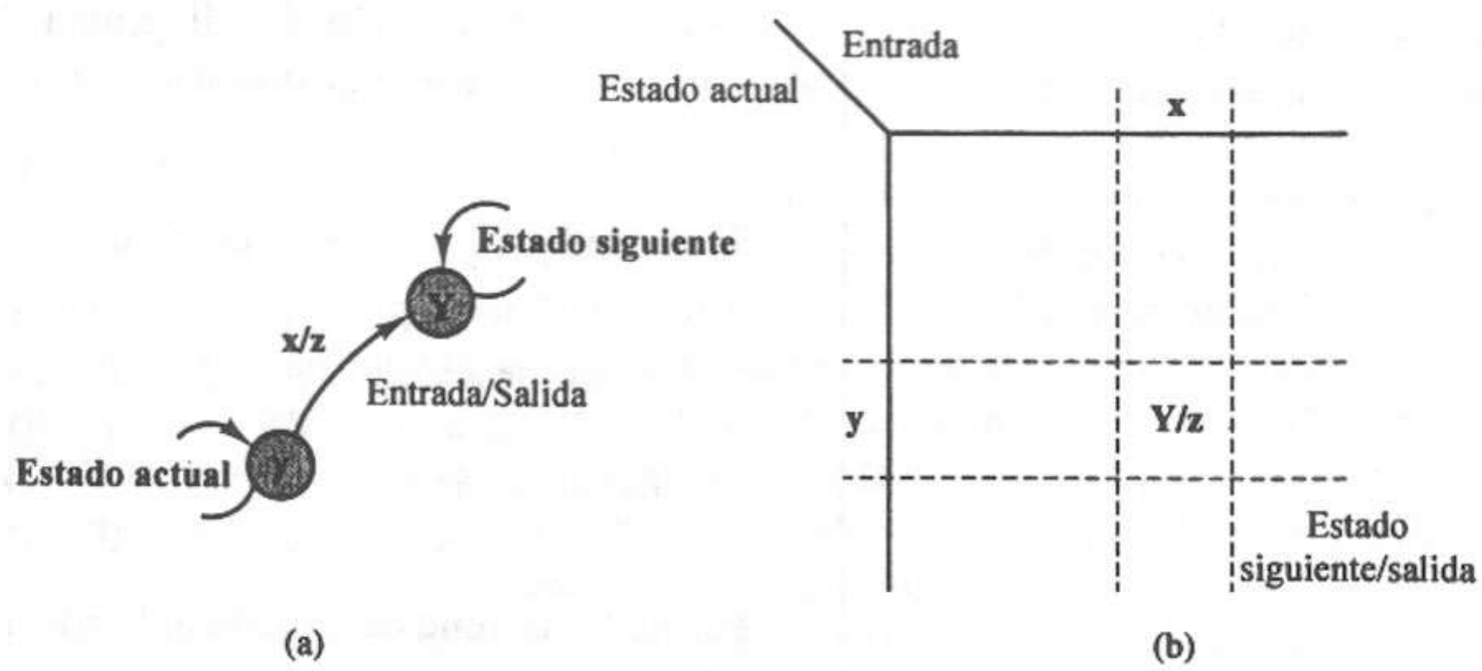


Figura 6.2 Tablas y diagramas de estados. **(a)** Diagrama de estados. **(b)** Tabla de estados.

En la práctica, los diagramas y las tablas de estados se etiquetan por lo general se rotulan con símbolos en vez de vectores. Por ejemplo, consideremos un circuito secuencial con dos variables de estado presente y_1 y y_2. Entonces

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Por tanto, el vector **y** puede tener cualquiera de cuatro valores posibles:

$$\begin{aligned} \mathbf{y} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} = A, & \mathbf{y} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} = C \\ \mathbf{y} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B, & \mathbf{y} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} = D \end{aligned} \tag{6.8}$$

Así, el circuito secuencial sólo tiene cuatro estados posibles, que podemos llamar A, B, C y D. En general, si r representa el número de dispositivos de memoria en un circuito con Ns estados, estas dos cantidades se relacionan mediante la expresión

$$2^{r-1} < N_s \leq 2^r \tag{6.9}$$

Utilizaremos esta expresión en capítulos posteriores.

EJEMPLO 6.1

Tenemos un circuito secuencial con una variable de entrada x, dos variables de estado y_1 y y_2 y una variable de salida z.

Entradas:	$x = 0$
	$x = 1$
Estados:	$[y_1,y_2]=[00]\equiv A$
	$[y_1,y_2]=[01]\equiv B$
	$[y_1,y_2]=[10]\equiv C$
	$[y_1,y_2]=[11]\equiv D$
Salidas:	$z = 0$
	$z = 1$

La figura 6.3 es el diagrama de estados para este circuito secuencial. Supongamos ahora que el circuito se encuentra inicialmente en el estado A; si se aplica una entrada $x = 0$, el siguiente estado es D y la salida es $z = 0$. Podemos leer esta información del diagrama de estados o de la tabla de estados. Consideremos ahora la aplicación de la siguiente serie de entrada al circuito:

$$x = 0110101100$$

Si el estado inicial es A, el circuito se comportará de la manera siguiente:

Instante:	0	1	2	3	4	5	6	7	8	9	10
Estado actual:	A	D	B	A	D	B	B	A	C	C	C
Entrada:	0	1	1	0	1	0	1	1	0	0	
Estado siguiente:	D	B	A	D	B	B	A	C	C	C	
Salida:	0	1	0	0	1	1	0	1	1	1	

Por tanto, la serie de entrada aplicada a la máquina en el estado A produce la serie de salida

$$z = 0100110111$$

y deja al circuito en el estado final C.

Estado actual	Entrada x = 0	Entrada x = 1
A	$D/0$	$C/1$
B	$B/1$	$A/0$
C	$C/1$	$D/0$
D	$A/0$	$B/1$

(a)

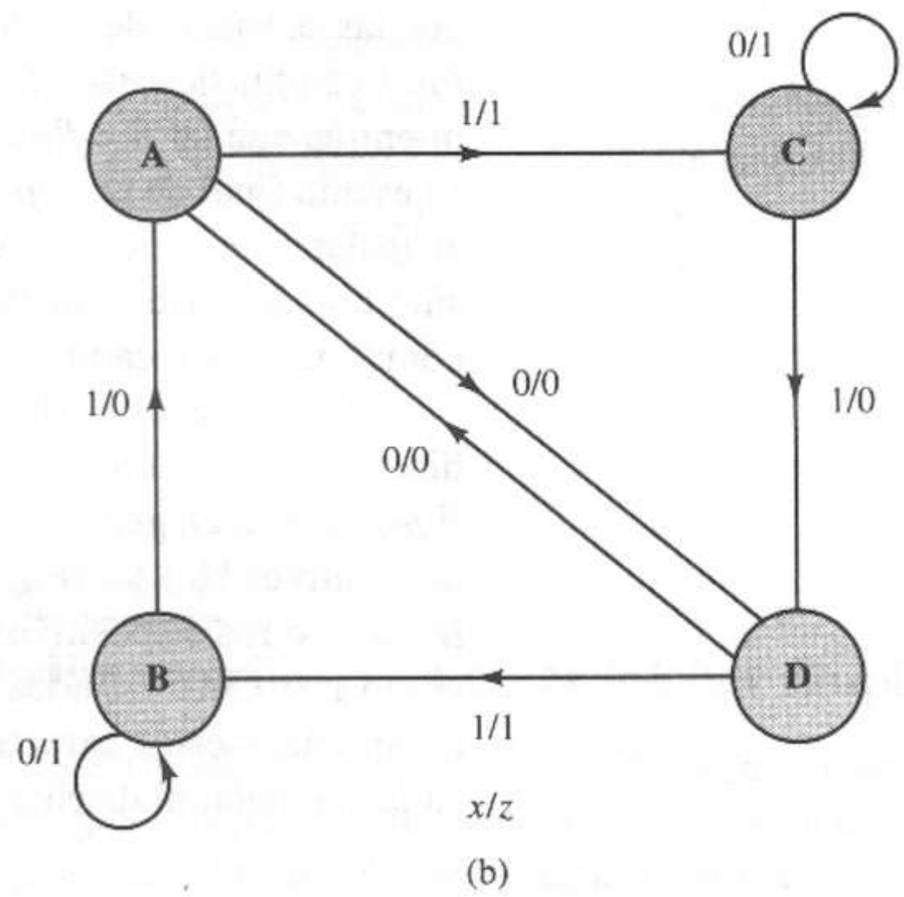


Figura 6.3 Ejemplo de un circuito secuencial. **(a)** Tabla de estados. **(b)** Diagrama de estados.

6.2 Dispositivos de memoria

Como hemos visto, una parte importante de una máquina secuencial es la unidad de memoria. Ahora centraremos nuestro análisis en las características externas de los dispositivos de memoria y no en sus funciones internas detalladas. En otras palabras, restringiremos nuestro análisis al uso de estos elementos en el diseño de sistemas digitales.

En las aplicaciones de circuitos de conmutación, casi todos los elementos de memoria son circuitos electrónicos *biestables*; es decir, existen indefinidamente en uno de dos posibles estados estables, 0 y 1. Los datos binarios se guardan en un elemento de memoria haciendo que el elemento pase al estado 0 para almacenar un 0 y al estado 1 para almacenar un 1. La salida Q del circuito indica el estado presente de la memoria. Cada circuito de memoria tiene una o más *entradas de excitación*, llamadas así por que sirven para "excitar" o llevar el circuito a un estado deseado. Los distintos dispositivos de memoria reciben por lo general su nombre con base en sus entradas de excitación particulares, diferentes de un dispositivo a otro.

Los dos tipos de elementos de memoria comúnmente utilizados en los circuitos de conmutación son los *latches* y los *flip-flops*. Un *latch* (*latch*) es un

elemento de memoria cuyas señales de entrada de excitación controlan el estado del dispositivo. Si un *latch* tiene una señal de entrada de excitación que obliga a la salida del dispositivo a asumir el valor 1, ésta es un *latch set*. Si el *latch* tiene una señal de entrada de excitación que obliga al dispositivo a asumir el valor 0, ésta es un *latch reset*. Si el dispositivo tiene señales de excitación *set* y *reset*, es un *latch set-reset*. La figura 6.4a muestra la operación de un *latch*.

Un *flip-flop* difiere de un *latch* por el hecho de que tiene una señal de control llamada *reloj*. La señal de reloj emite una instrucción al *flip-flop*, permitiéndole cambiar de estado de acuerdo con las señales de entrada de excitación. En los *latches* y los *flip-flops*, el siguiente estado queda determinado por las entradas de excitación. Sin embargo, como muestra la figura 6.4, un *latch* cambia de estado de inmediato, según sus señales de excitación de entrada, mientras que un *flip-flop* espera la señal de su reloj antes de cambiar de estado. El estado final de un *flip-flop* está determinado por sus valores de excitación en el instante en que se presenta la señal de reloj. De esta manera, se pueden sincronizar varios *flip-flops* de un circuito secuencial con una señal de reloj común para que cambien de estado todos juntos.

En el capítulo 10 presentaremos algunos procedimientos formales para diseñar circuitos con *flip-flops* y *latches*. En este capítulo describiremos circuitos *flip-flop* y *latch* genéricos, así como diversos módulos TTL que contienen estos dispositivos. La tabla 6.1 tiene la lista de los dispositivos que analizaremos. Muchos otros dispositivos están disponibles en el comercio. Hemos elegido éstos como introducción a las diversas características disponibles. El lector deberá comprender estas características para poder elegir un dispositivo adecuado a cada situación al diseñar circuitos secuenciales.

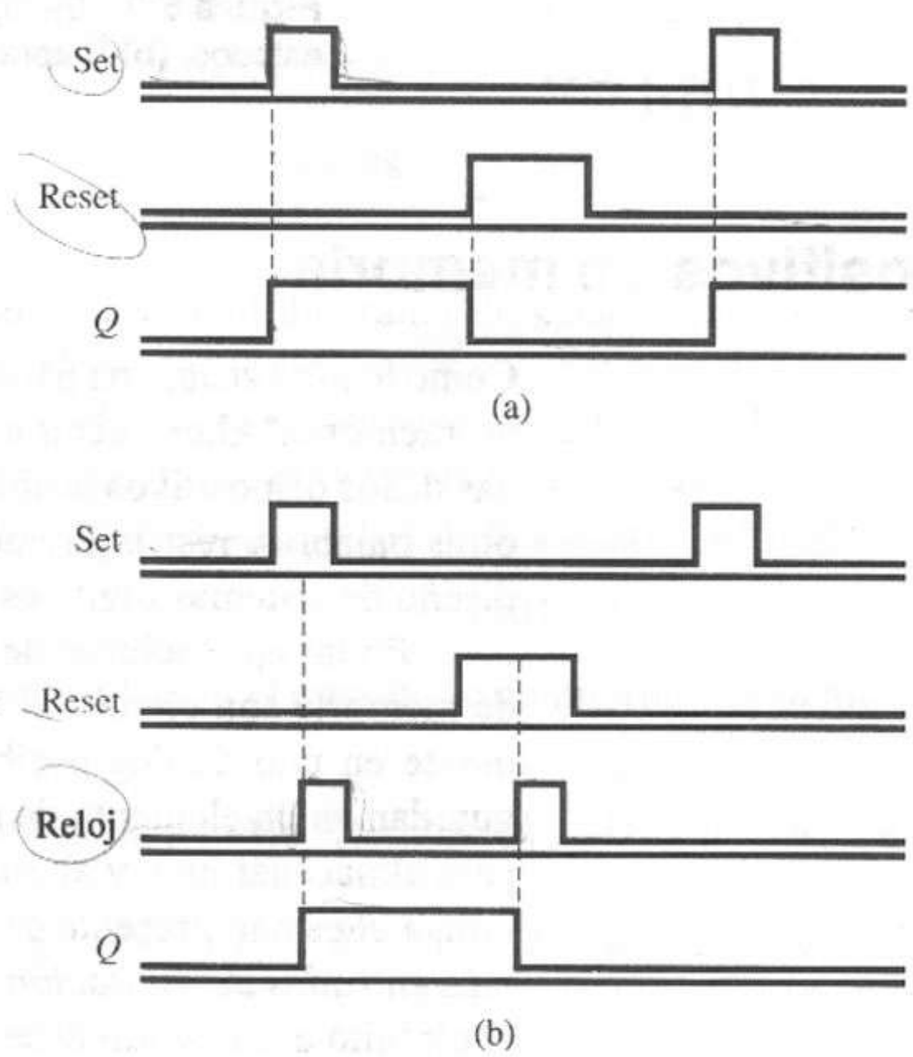


Figura 6.4 Tiempos en *latch* y *flip-flop*. **(a)** El *latch* responde de inmediato a la excitación. **(b)** El *flip-flop* responde sólo en una señal de reloj.

TABLA 6.1 ELEMENTOS DE MEMORIA TTL [1]

Dispositivo	Número de elementos	Descripción del elemento
74LS73A	2	*Flip-flop* JK disparado por transición negativa con *clear*
7474	2	*Flip-flop* D disparado por transición positiva con *preset* y *clear*
74LS75	4	*Latch* D con activación
7476	2	*Flip-flop* JK activado por pulsos con *preset* y *clear*
74111	2	*Flip-flop* JK maestro-esclavo con *preset*, *clear* y bloqueo de datos
74116	2	*Latch* D libre de riesgos de 4 bits con *clear* y activación dual
74175	4	*Flip-flop* D disparado por transición positiva con *clear*
74273	8	*Flip-flop* D disparado por transición positiva con *clear*
74276	4	*Flip-flop* J$\overline{K}$ disparado por transición negativa con *preset* y *clear*
74279	4	*Latch* SR con entradas activas bajas

6.3 *Latches*

6.3.1 *Latch set-reset*

Uso de retroalimentación para crear *latches* simples

Considere la compuerta OR de la figura 6.5a. Suponga que ambas entradas se encuentran en 0 lógico. Si conectamos la salida de nuevo a una de las entradas, como en la figura 6.5b, la compuerta se estabiliza con una salida de 0. Si aplicamos un 1 lógico a la entrada no conectada *S*, como en la figura 6.5c, ¿qué le sucede a la salida de la compuerta OR? Cambia a 1 lógico. Así, hemos establecido (*set*) la salida del dispositivo *Q* como 1 lógico. Si cambiamos la entrada *S* de nuevo a 0 lógico, la salida *Q* sigue con el valor 1 lógico, debido a la retroalimentación a la otra entrada de la compuerta OR, como en la figura 6.5d. Así, este dispositivo asume el valor 1 lógico de manera permanente y lo llamamos un *latch set*.

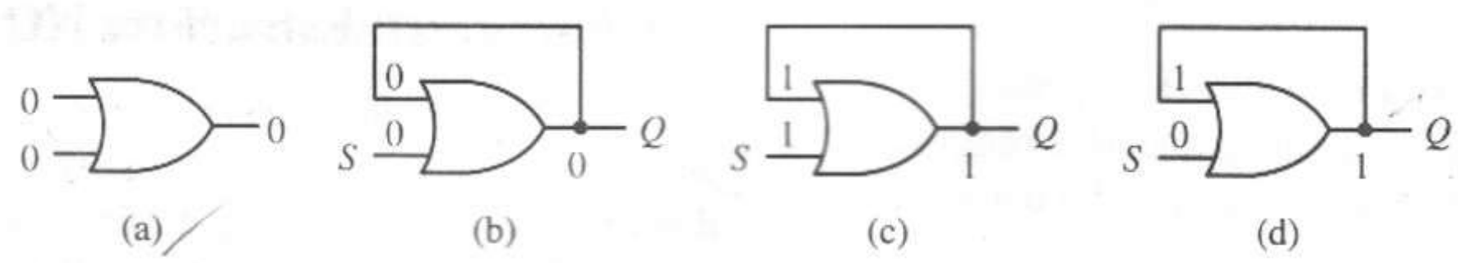


Figura 6.5 *Latch set.* **(a)** Compuerta OR. **(b)** Con retroalimentación. **(c)** Salida con valor 1. **(d)** La retroalimentación mantiene $Q = 1$.

Reemplacemos la compuerta OR de la figura 6.5 por compuertas NOR y NOT en cascada, como en la figura 6.6a. Éste es un circuito de *latch set* equivalente. Si utilizamos la salida de la compuerta NOR como salida *Q* del *latch*, generamos el circuito de la figura 6.6b. Examine el funcionamiento de estas compuertas en cascada. La salida de la compuerta NOR se encuentra inicialmente en 1 lógico si sus dos entradas tienen el valor 0 lógico. La señal de

retroalimentación de la compuerta NOT es 0 lógico, lo que produce una condición estable. Al colocar un 1 lógico en la entrada no conectada *R* de la compuerta NOR, su salida debe ser 0 lógico, como se muestra en la figura 6.6c, lo que hace que la señal de retroalimentación sea 1 lógico. Así, la salida del *latch Q* se restablece (*reset*) a 0 lógico. Si *R* vuelve a asumir el valor 0 lógico, como se muestra en la figura 6.6d, no habrá cambios en la salida, debido a la señal de retroalimentación. Por tanto, la salida *Q* mantendrá el valor 0 lógico, ignorando los cambios posteriores en la entrada *R*. Este circuito se llama un *latch reset*. La figura 6.6e presenta otra vista del circuito.

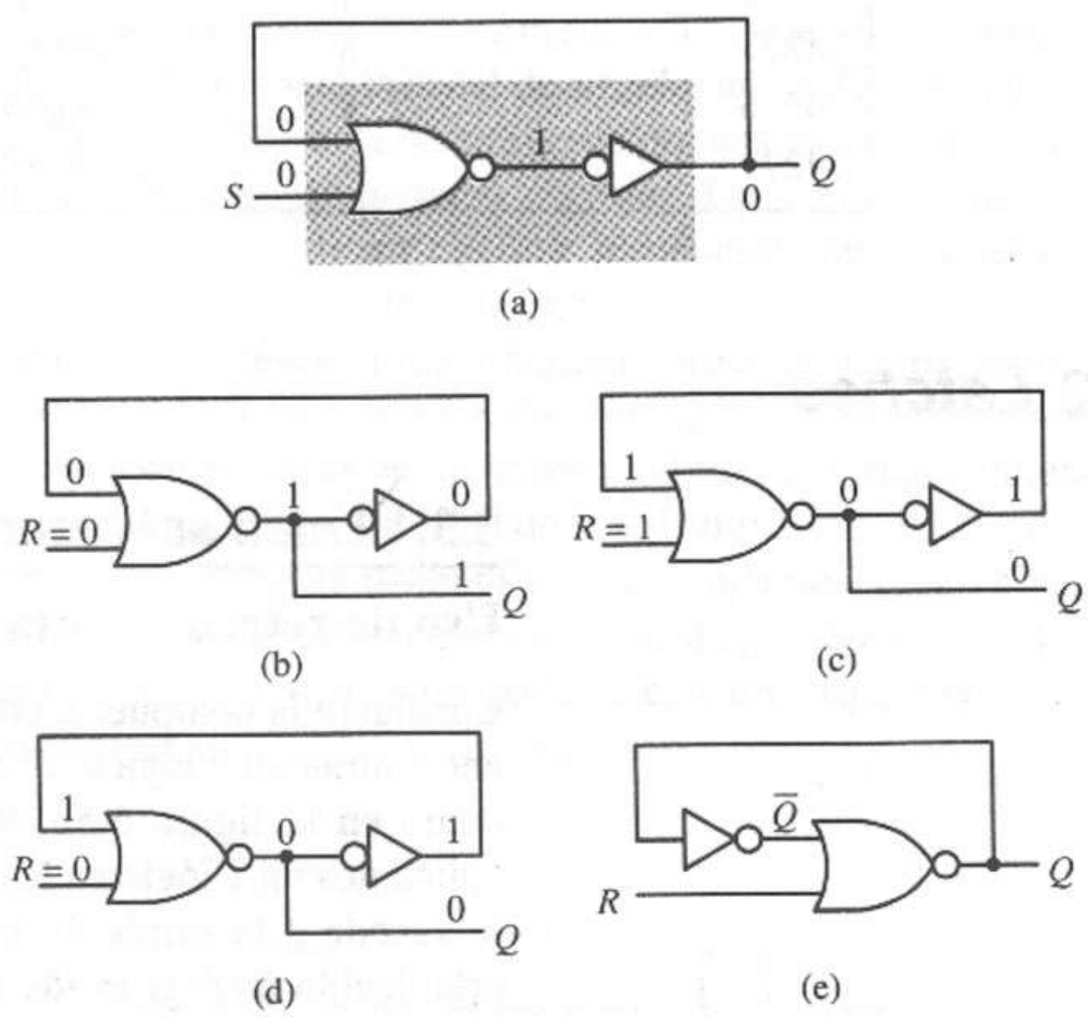


Figura 6.6 *Latch reset.* **(a)** *Latch set* redibujada. **(b)** *Latch reset* estable con $Q = 1$. **(c)** $R = 1$ reestablece la *latch* como $Q = 0$. **(d)** Los cambios posteriores son inhibidos. **(e)** Vista alternativa del *latch reset.*

Latch set-reset: Estructura NOR

Los dispositivos que se estabilizan en un estado lógico no son muy útiles, excepto en situaciones de diseño poco usuales. Si combinamos las características de los dos *latches* ya descritos en un único circuito, podemos activar o desactivar el circuito *latch* según sea necesario. Remplacemos la compuerta NOT del circuito de la figura 6.6a por una compuerta NOR de dos entradas, *N*2, conectadas a modo de operar como una compuerta NOT, como en la figura 6.7a. El dispositivo aún opera como *latch set*. Ahora bien, si desconectamos la entrada inferior de la compuerta NOR *N*2 como en la figura 6.7b, esta entrada funcionará como una excitación *reset* para el dispositivo (la misma función que en la figura 6.6e). Así, hemos creado un *latch set-reset* (*latch SR*) con dos compuertas NOR de dos entradas. El diagrama tradicional del circuito es la forma con acoplamiento cruzado de la figura 6.7c. Adoptaremos el símbolo lógico de la figura 6.7d para representar el *latch* SR. Lo utilizaremos como bloque de construcción para otros circuitos en esta sección.

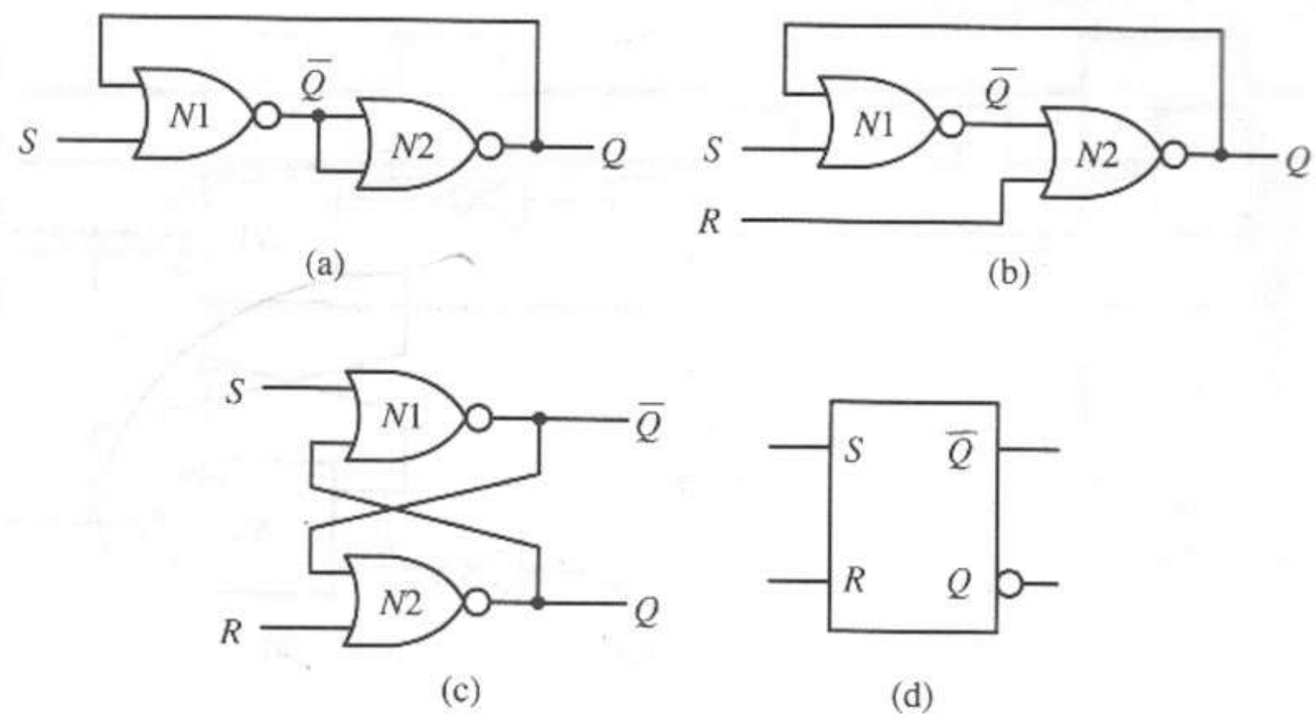


Figura 6.7 *Latch set-reset* (*Latch* SR). **(a)** *Latch* S redibujada. **(b)** *Latch* SR. **(c)** Vista tradicional del *latch* SR. **(d)** Símbolo lógico del *latch* SR.

Latch SR: Estructura NAND

¿Podemos obtener los mismos dispositivos funcionales que dedujimos con compuertas NOR utilizando compuertas NAND? Consideremos las compuertas NAND con acoplamiento cruzado de la figura 6.8a. Si las dos entradas S y R asumen el valor 0 lógico, las compuertas NOT aplican señales 1 lógicas a las compuertas NAND con acoplamiento cruzado. Pero si una de las entradas de una compuerta NAND de dos entradas se coloca en 1 lógico, actúa como una compuerta NOT; es decir,

$$\text{Compuerta } N1\,(S=0): \quad \overline{\bar{S}\cdot\bar{Q}} = \overline{1\cdot\bar{Q}} = \bar{\bar{Q}} = Q$$

$$\text{Compuerta } N2\,(R=0): \quad \overline{\bar{R}\cdot Q} = \overline{1\cdot Q} = \bar{Q}$$

Así, las compuertas con acoplamiento cruzado de la figura 6.8 asumen la función de un par de compuertas NOT cuando las entradas son $S = R = 0$, como se muestra en la figura 6.8b. (Indicamos las compuertas NOT con líneas punteadas dentro de las compuertas NAND para ilustrar que éstas últimas realmente actúan como inversores). El bucle de compuertas NOT de la figura 6.8b forma la celda de almacenamiento biestable para el *latch*. La salida Q se introduce en una compuerta NOT para generar $\overline{Q}$, y $\overline{Q}$ se introduce en la segunda compuerta NOT para regenerar Q.

Si alguna de las entradas S o R está activa, es decir, con el valor 1 lógico, y la otra tiene el valor 0 lógico, una parte de la configuración con acoplamiento cruzado simétrico se altera cuando se aplica un 0 lógico a la entrada de una de las compuertas NAND; es decir,

$$\text{Compuerta } N1\,(S=1): \quad \overline{\bar{S}\cdot\bar{Q}} = \overline{0\cdot\bar{Q}} = \bar{0} = 1$$

o bien

$$\text{Compuerta } N2\,(R=1): \quad \overline{\bar{R}\cdot Q} = \overline{0\cdot Q} = \bar{0} = 1$$

Así, una entrada 0 lógica a la compuerta NAND hace que su salida asuma el valor 1 lógico. El 1 lógico en la salida de la compuerta NAND activa hace que la salida de la otra NAND del par asuma el valor 0 lógico, ya que

$$\text{Compuerta } N2\,(S=1,\ R=0): \quad \bar{Q} = \overline{\bar{R}\cdot Q} = \overline{\bar{0}\cdot 1} = \overline{1\cdot 1} = \bar{1} = 0$$

o bien

$$\text{Compuerta } N1\,(S=0,\ R=1): \quad Q = \overline{\bar{S}\cdot\bar{Q}} = \overline{\bar{0}\cdot 1} = \overline{1\cdot 1} = \bar{1} = 0$$

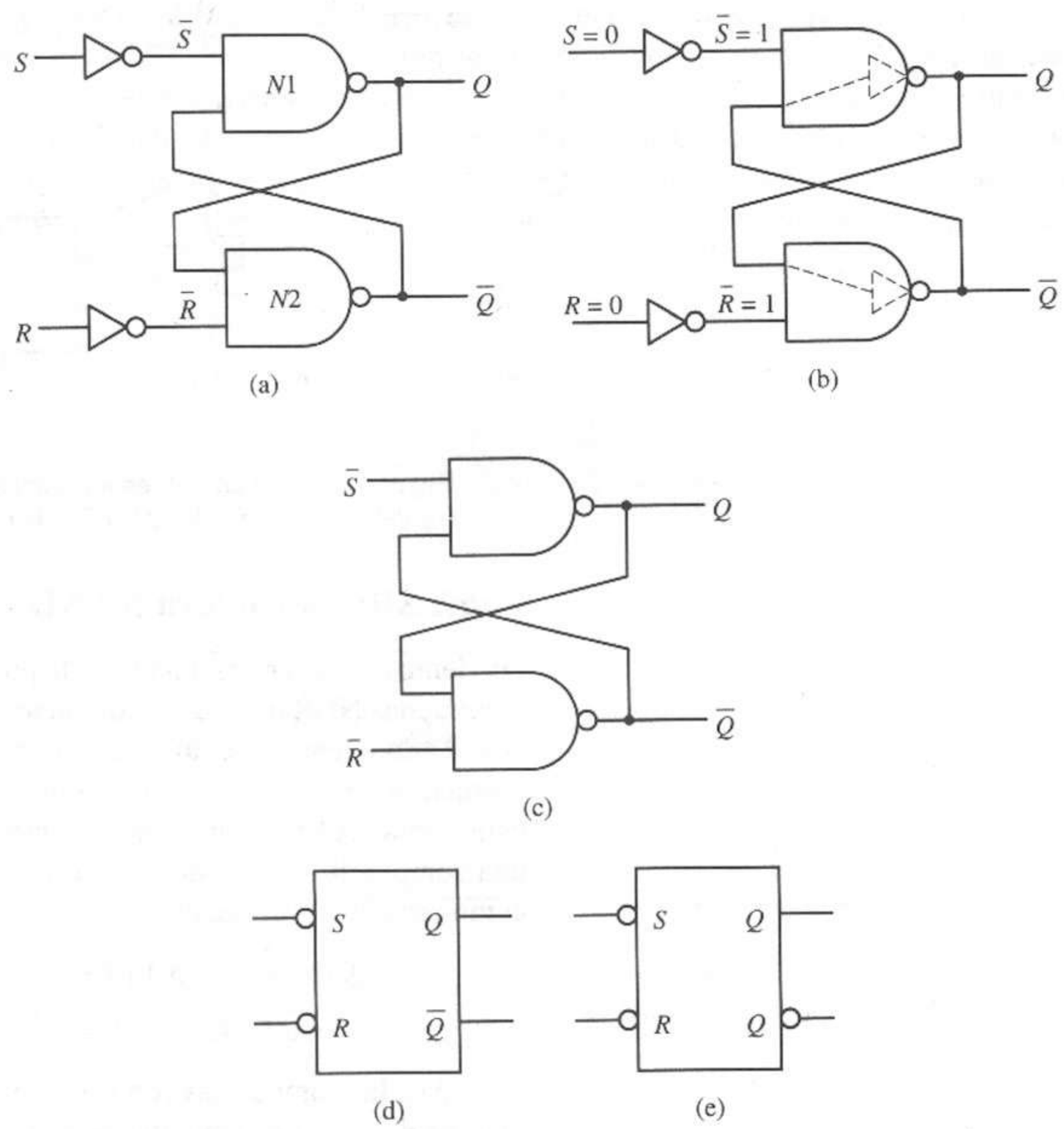


Figura 6.8 *Latch* SR NAND. **(a)** Diagrama lógico. **(b)** Modo de almacenamiento. **(c)** Lógica reducida. **(d)** Símbolo lógico 1. **(e)** Símbolo lógico 2.

Así, un 1 lógico en S hace que Q asuma el valor 1 lógico, por lo que $\overline{Q}$ asume el valor 0 lógico. En otras palabras, un 1 lógico en S *establece* el *latch*. De la misma manera, un 1 lógico en la entrada R hace que $\overline{Q}$ asuma el valor 1 lógico, por lo que la salida asume el valor 0 lógico, *restableciendo* el *latch*.

Eliminemos las dos compuertas NOT de las entradas del circuito, como en la figura 6.8c. Las entradas del dispositivo se convierten en $\overline{S}$ y $\overline{R}$. En otras palabras, las entradas del *latch* son activas bajas (normalmente, altas en el estado activo y pasan a bajas para activar el dispositivo). Adoptaremos los símbolos lógicos de las figuras 6.8d y e para representar un *latch* SR con entradas activas bajas.

Diagramas de tiempos y parámetros de retardo en un *latch SR*

Podemos describir el funcionamiento de cualquier *latch* mediante un diagrama de tiempos. La figura 6.9a muestra la acción inducida en el *latch* NOR con

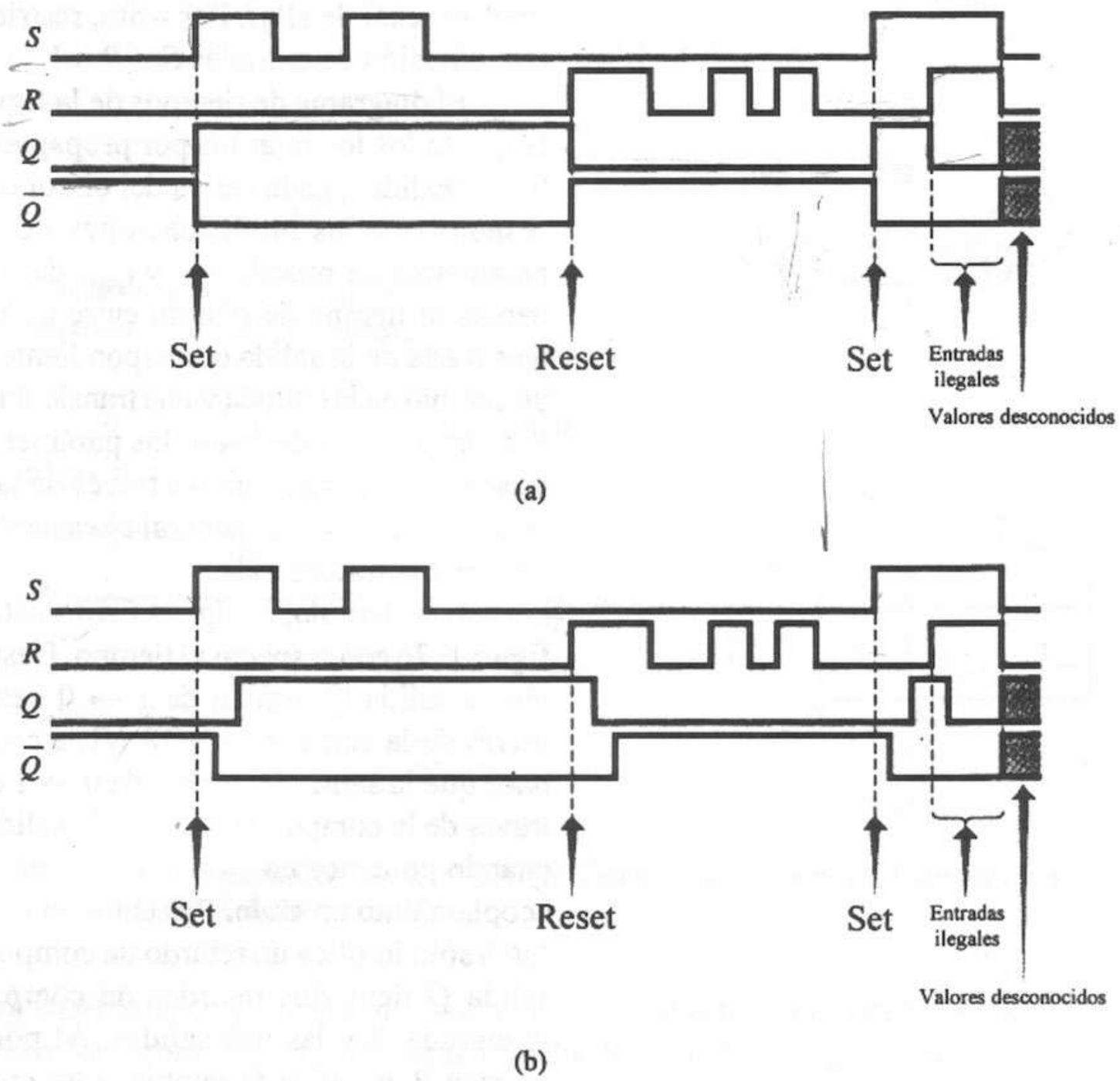


Figura 6.9 Diagrama de tiempos del *latch set-reset*. **(a)** Caso ideal (retardo por compuertas nulo). **(b)** Tiempos reales con retardos de compuerta distintos de cero.

acoplamiento cruzado de la figura 6.7c al aplicar secuencias de pulsos a las entradas *set* y *reset*, S y R. Elegimos las formas de onda para S y R a modo de ilustrar varias características del *latch*. Supondremos que el estado inicial del *latch* es 0 lógico ($Q = 0$). El dispositivo responde al primero de una serie de pulsos sobre una de sus entradas, pero ignora los demás pulsos hasta que un pulso interviene en la otra entrada.

Observe que al colocar las señales con el valor 1 lógico en ambas entradas R y S, las salidas Q y $\overline{Q}$ asumen el valor 0 lógico. Cuando las dos entradas vuelven al valor 0 lógico, se crea una *condición de competencia* y, por tanto, no podemos determinar con certeza el estado que asumirá el dispositivo. En el mundo real, es virtualmente imposible que dos eventos sucedan exactamente al mismo tiempo, ¡aunque así lo desee! Si la señal R regresa al 0 lógico antes que S, el estado final de Q será 1 lógico. Si S regresa primero al 0 lógico, el dispositivo asume el valor 0 lógico. Si R y S regresan al 0 lógico al mismo tiempo, o casi al mismo tiempo, las dos compuertas NOR competirán para ganar el control de la salida Q. Si ambas son equivalentes desde el punto de vista eléctrico, ¡la salida oscilará! En los circuitos prácticos, gana una de las compuertas, pero no podemos

predecir cuál de ellas. Por tanto, restringimos el uso del *latch* SR excluyendo la combinación de entradas $S = R = 1$.

El diagrama de tiempos de la figura 6.9a representa una situación ideal en la que todos los retardos por propagación en la compuerta se consideran como 0. En realidad, cada salida del circuito requiere un tiempo distinto de cero para responder a las modificaciones de sus entradas, como lo especifican los parámetros de retardo t_{PLH} y t_{PHL}, definidos en el capítulo 2. Recuerde que t_{PLH} denota al tiempo de retardo entre un cambio en la entrada y una transición de baja a alta de la salida correspondiente. De igual manera, t_{PHL} es el retardo entre un cambio en la entrada y una transición de alta a baja de la salida correspondiente. Para un circuito de *latch*, los parámetros t_{PLH} y t_{PHL} representan la suma de los retardos por propagación a través de las compuertas entre la entrada y salida de un *latch* dado; por lo general se especifican parámetros de retardo distintos para cada par entrada/salida.

Por ejemplo, la figura 6.10 ilustra el comportamiento del *latch* SR de la figura 6.7c con respecto al tiempo. Después de un cambio en S de $0 \rightarrow 1$, observe que la salida $\overline{Q}$ cambia de $1 \rightarrow 0$ después del retardo por propagación t_{PHL} a través de la compuerta NOR $N1$; a continuación, la señal de retroalimentación hace que la salida Q cambie de $0 \rightarrow 1$ después del tiempo de propagación t_{PLH} a través de la compuerta $N2$. Así, la salida $\overline{Q}$ siempre cambia entes que la salida Q cuando ponemos en 1 un *latch* SR construida a partir de compuertas NOR con acoplamiento cruzado. Por tanto, el tiempo t_{PHL} de la entrada S a la salida $\overline{Q}$ del *latch* sólo implica un retardo de compuerta, mientras que t_{PLH} de la entrada S a la salida Q tiene dos retardos de compuerta. Existe una relación similar entre la entrada R y las dos salidas. Al poner en cero el *latch* con un pulso en la entrada R, la salida Q cambia antes que la salida $\overline{Q}$, como en la figura 6.10. En vista de estos retardos por propagación distintos de cero a través de las compuertas del circuito *latch*, la figura 6.9b presenta una imagen del funcionamiento del *latch* más real que el diagrama de tiempos ideal de la figura 6.9a.

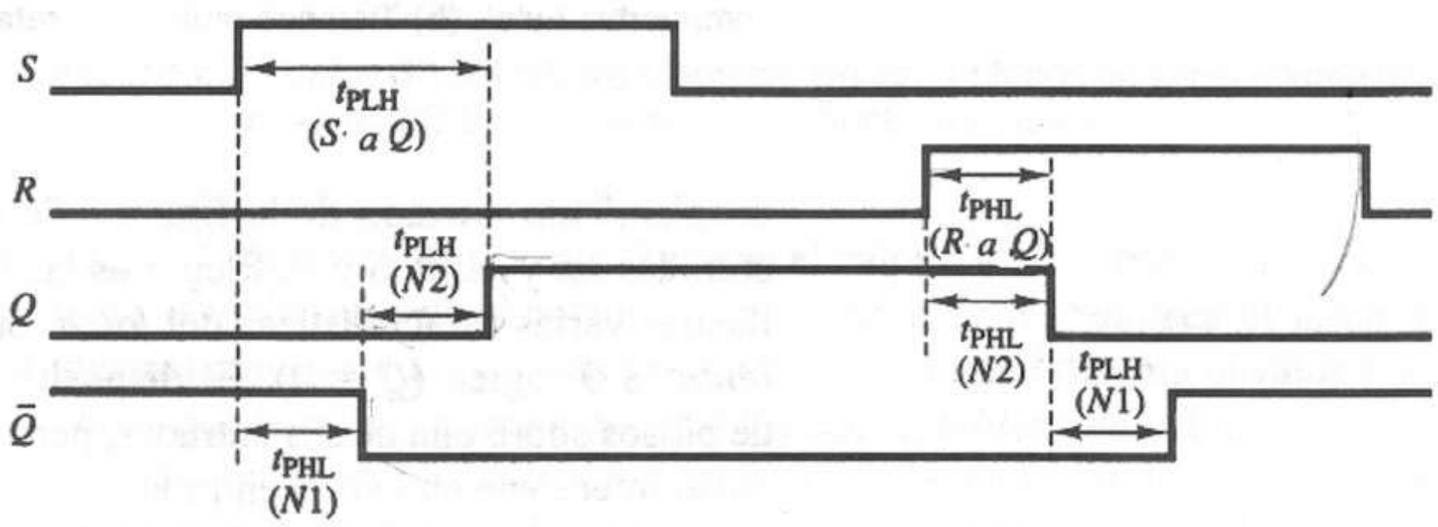


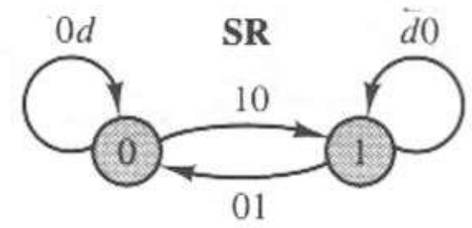
Figura 6.10 Retardos de propagación del *latch* SR.

Tabla de excitación y ecuación característica del *latch* SR

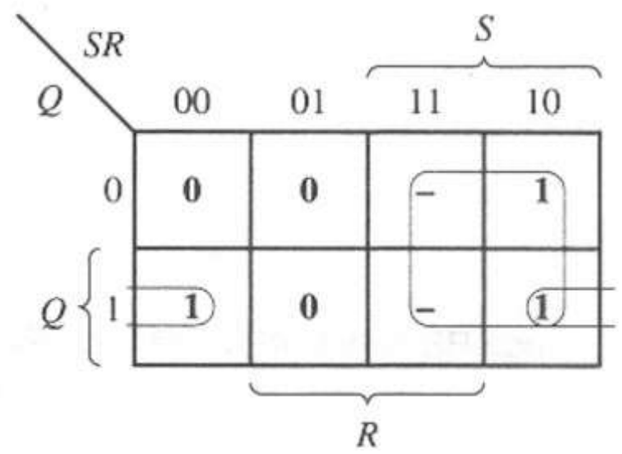
La *tabla de excitación* de la figura 6.11a resume el funcionamiento lógico del *latch* SR. La tabla de excitación es simplemente la tabla de estados del *latch* y muestra las transiciones de estado para cada combinación de las entradas de excitación. Las columnas S y R son las entradas aplicadas a un *latch* SR mientras

Entradas de excitación S	R	Estado actual Q	Estado siguiente Q^*	
0	0	0	0	Ningún cambio
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	×	No permitido
1	1	1	×	

(a)



(b)



(c)

Figura 6.11 Características del *latch* SR. **(a)** Tabla de excitación. **(b)** Diagrama de estados. **(c)** Mapa K de la salida Q^*.

está en el estado Q. La columna rotulada Q es el estado del *latch* SR *antes* de aplicar una combinación de entradas a S y R. La columna Q^* es el estado del *latch* SR *después* de aplicar las entradas SR y obtener un resultado estacionario. La columna Q es el *estado presente* del *latch* SR y la columna Q^* es el *siguiente estado*.

Podemos representar la información de la figura 6.11 como un diagrama de estados (Fig. 6.11b) o como un mapa K (Fig. 6.11c), donde damos el valor del siguiente estado Q^* en función de las entradas, S y R, y del estado presente Q. El mapa K nos permite deducir la siguiente expresión lógica para Q^*, llamada *ecuación característica* del *latch* SR:

$$Q^* = S + \overline{R}Q \tag{6.10}$$

La ecuación característica recibe ese nombre porque caracteriza el funcionamiento del *latch*. Por ejemplo, podemos clasificar el funcionamiento del *latch* en tres casos.

Caso 1: $S = R = 0$. La ecuación 6.10 se reduce a $Q^* = Q$, lo que indica que el estado no cambia.

Caso 2: $S = 1, R = 0$. La ecuación 6.10 se reduce a $Q^* = 1$, lo que representa la operación de establecimiento (*set*).

Caso 3: $S = 0, R = 1$. La ecuación 6.10 reduce a $Q^* = 0$, representando la operación de restablecimiento (*reset*).

Módulo *latch* SR 74279 cuádruple

Hay *latches* SR disponibles en forma comercial, como el módulo SN74279 [1] citado en la tabla 6.1. El módulo 74279 tiene cuatro unidades de *latch*, dos como se ilustra en la figura 6.8 y dos como se muestra en la figura 6.12a. Cada uno de los *latches* del segundo par incluye una entrada $\overline{S}$ adicional, de modo que

las dos entradas $\overline{S_1}$ y $\overline{S_2}$ bajas activas se conectan a la misma compuerta NAND, como se muestra en la figura 6.12b, para que al activar cualquiera de ellas se active el *latch*. Para minimizar el tamaño global del módulo, sólo las salidas verdaderas, Q, se conectan a *pines* externos del paquete.

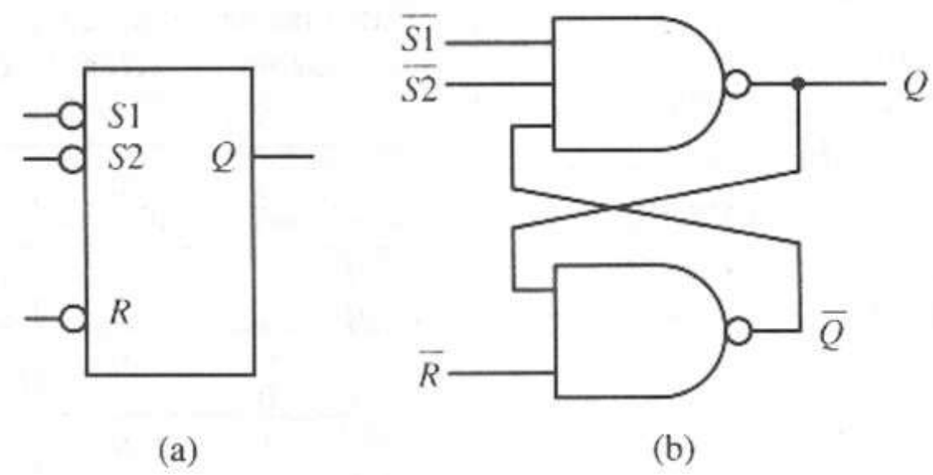


Figura 6.12 *Latch* 74279 con dos entradas *set*. **(a)** Símbolo lógico. **(b)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

Los retardos por propagación típicos del 74279 son 12 ns para t_{PLH} desde cualquier entrada $\overline{S}$ hasta la salida Q y 15 ns desde la entrada $\overline{R}$ hasta la salida Q [1]. Observe que el cambio en R debe propagarse por dos compuertas para afectar la salida y, por tanto, exhibe un retardo por propagación mayor que los cambios debidos a $\overline{S}$.

6.3.2 *Latch* SR con compuertas

Con frecuencia es deseable utilizar una señal especial de control para inhibir los cambios de estado de un *latch* SR mientras S y R cambian. Cuando S y R están listas, activamos esta señal de control para habilitar el *latch* y que responda a los nuevos valores de S y R. Este dispositivo se conoce como *latch SR con compuertas*, pues podemos pensar que la señal de control abre una compuerta a través de la cual se propagan hasta la salida las señales sobre las entradas S y R.

Estructura de circuito

En la figura 6.13a agregamos una señal de control C a un *latch* SR para aplicar las entradas S y R. Las dos compuertas AND aplican las señales de control S y R durante los intervalos de tiempo en los que la señal de activación C es alta (1 lógico). Cuando C asume el valor 0 lógico, las entradas del *latch* SR se mantienen en el estado S=R=0 (sin cambio). Así, el funcionamiento del *latch* es el siguiente: cuando $C = 0$, no ocurren cambios y, por tanto, el dispositivo es estable; cuando $C = 1$, la tabla de excitación (figura 6.11a) y la ecuación característica (ecuación 6.10) del *latch* SR describen la función de éste.

Si cambiamos las compuertas AND por compuertas NAND y utilizamos compuertas NAND con acoplamiento cruzado para el *latch* SR, obtenemos el circuito de la figura 6.13b. Al sustituir el circuito de la figura 6.8c por el símbolo de *latch* de la figura 6.13b obtenemos la implantación mediante compuertas NAND del *latch* SR con compuertas que aparecen en la figura 6.13c. El símbolo lógico genérico para el *latch* SR con compuertas aparece en la figura 6.13d.

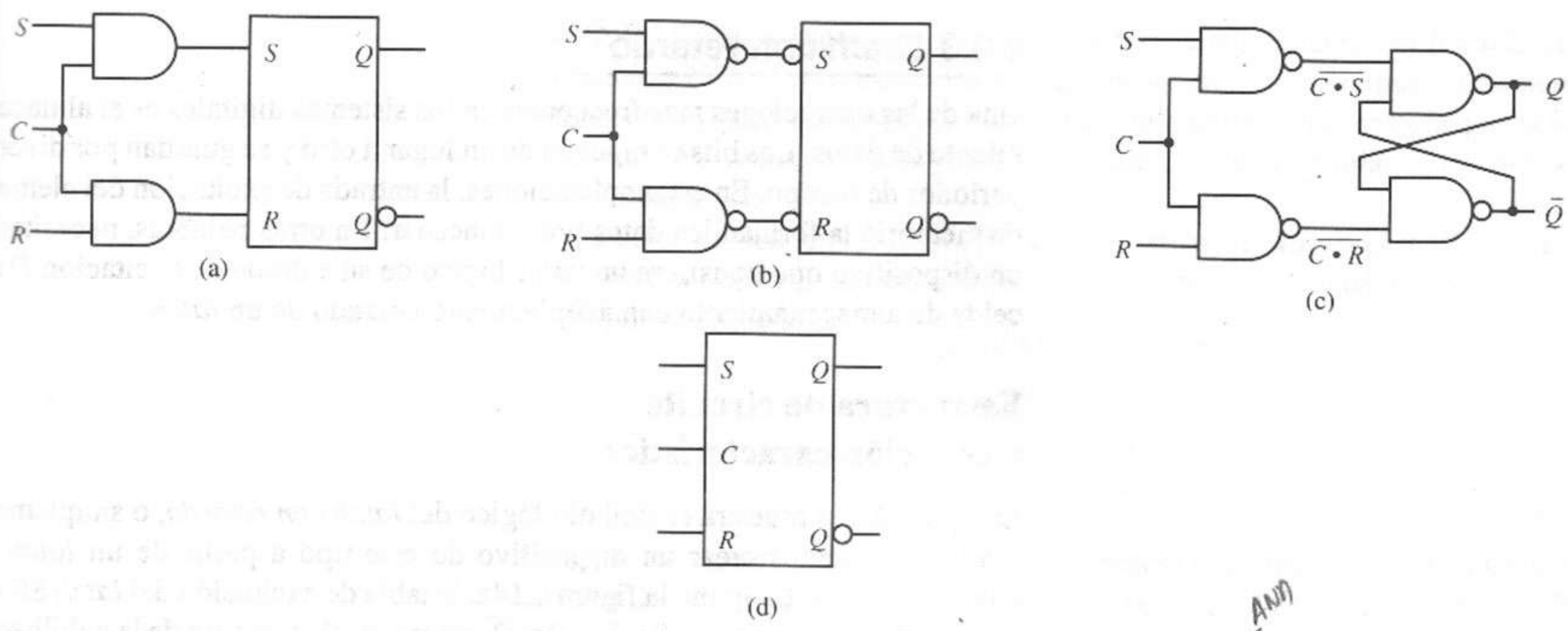
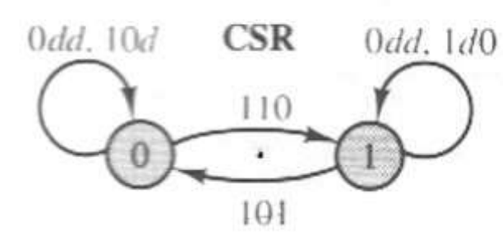


Figura 6.13 *Latch* SR con compuertas. **(a)** *Latch* SR con ~~NOR~~ AND. **(b)** *Latch* SR con NAND. **(c)** Diagrama lógico NAND. **(d)** Símbolo lógico.

Entradas de activación	Entradas de excitación		Estado actual	Estado siguiente	
C	S	R	Q	Q^*	
0	×	×	0	0	Retención
0	×	×	1	1	
1	0	0	0	0	Sin cambio
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	×	No permitido
1	1	1	1	×	

(a)



(b)

Figura 6.14 Características del *latch* SR con compuertas. **(a)** Tabla de excitación. **(b)** Diagrama de estados.

Ecuación característica

Las figuras 6.14a y b muestran la tabla de excitación completa y el diagrama de estados del *latch* S con compuertas. De la tabla de excitación podemos deducir la siguiente ecuación característica para el *latch* SR con compuertas:

$$Q^* = SC + \overline{R}Q + \overline{C}Q \tag{6.11}$$

Observe que si $C = 0$, la ecuación 6.11 se reduce a $Q^* = Q$, lo que significa que el estado presente se *mantiene*. Al sustituir $C = 1$ en la ecuación 6.11 obtenemos $Q^* = S + \overline{R}Q$, la ecuación característica del *latch* SR simple y, por tanto, el *latch* está habilitado.

6.3.3 *Latch* con retardo

Una de las operaciones más frecuentes en los sistemas digitales es el almacenamiento de datos. Los bits se mueven de un lugar a otro y se guardan por diversos periodos de tiempo. En estas aplicaciones, la entrada de excitación del elemento de memoria la forman los datos por almacenar. En otras palabras, necesitamos un dispositivo que transfiera un valor lógico de su entrada de excitación D a la celda de almacenamiento con acoplamiento cruzado de un *latch*.

Estructura de circuito y ecuación característica

La figura 6.15a muestra el símbolo lógico del *latch con retardo*, o simplemente *latch D*. Podemos crear un dispositivo de este tipo a partir de un *latch* SR con compuertas. Examine la figura 6.14a, la tabla de excitación del *latch* SR con compuertas. Si asignamos $S = D$ y $R = \overline{D}$, entonces, al momento de la habilitación restringimos el funcionamiento del *latch* a las cuatro filas de la figura 6.14a en las que $S = 1$ y $R = 0$, la condición *set* o $S = 0$ y $R = 1$, la condición *reset*. Por tanto, podemos reducir la tabla de excitación del *latch* SR con compuertas a la tabla de excitación del *latch* D de la figura 6.16a. La figura 6.16b muestra el diagrama de estados correspondiente.

La figura 6.16b muestra la implantación NAND del *latch* D. Observe que este circuito es simplemente un *latch* SR con compuertas y entradas $S = D$ y

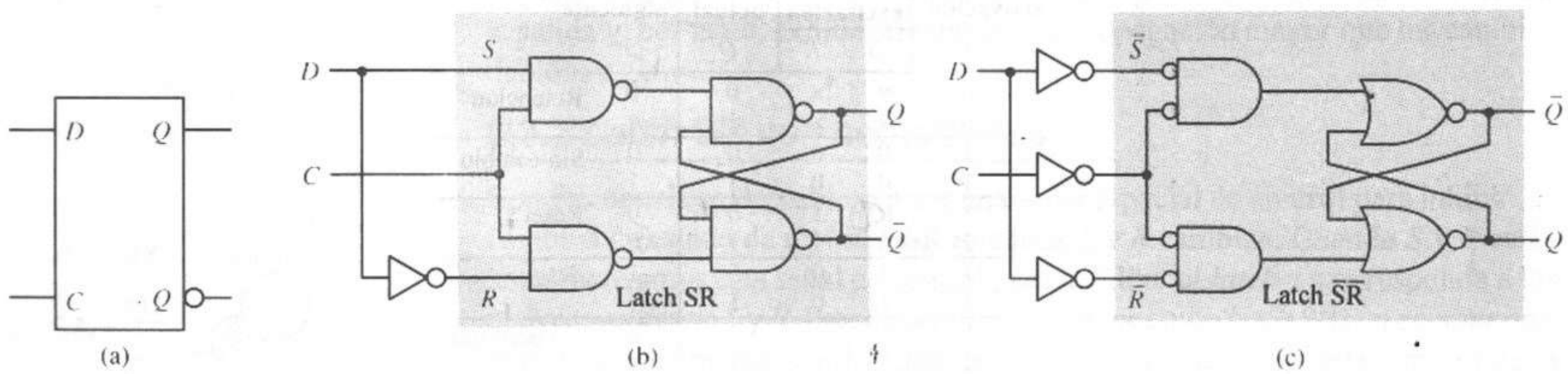
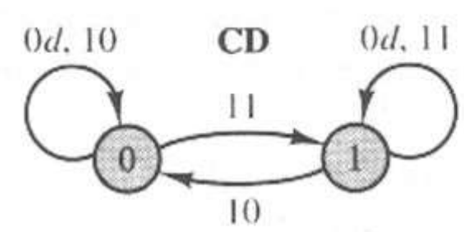


Figura 6.15 *Latch* con retardo (*latch* D). **(a)** Símbolo lógico. **(b)** Implantación NAND. **(c)** Implantación NOR.

Entrada de activación C	Entrada de excitación D	Estado actual Q	Estado siguiente Q^*	
0	×	0	0	Retención
0	×	1	1	
1	0	0	0	Almacenar 0
1	0	1	0	
1	1	0	1	Almacenar 1
1	1	1	1	

(a)



(b)

Figura 6.16 Características del *latch* D. **(a)** Tabla de excitación. **(b)** Diagrama de estados.

$R = \overline{D}$. La figura 6.15c muestra una implantación equivalente del *latch* D con compuertas NOR. En este circuito utilizamos un *latch* $\overline{SR}$, con inversores en las entradas D y C para que ambas entradas sean altas activas. Por tanto, la operación funcional de esta implantación NOR es idéntica a la de la implantación NAND de la figura 6.15b.

Podemos deducir la ecuación característica del *latch* D a partir de la ecuación del *latch* SR con compuertas, sustituyendo en la ecuación 6.11 D por S y $\overline{D}$ por R, como sigue:

$$\begin{aligned} Q^* &= SC + \bar{R}Q + \bar{C}Q \\ &= DC + (\bar{\bar{D}})Q + \bar{C}Q \\ &= DC + DQ + \bar{C}Q \\ &= DC + \bar{C}Q \end{aligned} \tag{6.12}$$

La ecuación característica 6.12 describe el funcionamiento del *latch* D. Cuando la señal de activación es baja, ($C = 0$), la ecuación 6.12 se reduce a $Q^* = Q$. En este caso, el *latch* pasa al modo operativo de *retención*; es decir, no hay cambio alguno, y mantiene el último valor introducido de D. En otras palabras, cuando $C = 0$, se mantienen los datos (se almacenan) en el *latch*. Al sustituir $C = 1$ en la ecuación 6.12 obtenemos $Q^* = D$. Así, el siguiente estado Q^* debe ser el valor de la entrada de excitación D si la señal de activación C es alta; es decir, la entrada de excitación D se conecta de manera directa a la salida Q. En este caso, el *latch* D está en el modo con *compuertas* o *habilitado*.

El diagrama de tiempos de la figura 6.17 muestra el funcionamiento del *latch* D. Observe que, cuando C es alta, cualquier cambio en D pasa por la salida del *latch*. Así, el *latch* guarda el último valor de D presente en su entrada cuando la señal de activación pasa de alta a baja, es decir, cuando hay una transición 1 → 0 en la señal C.

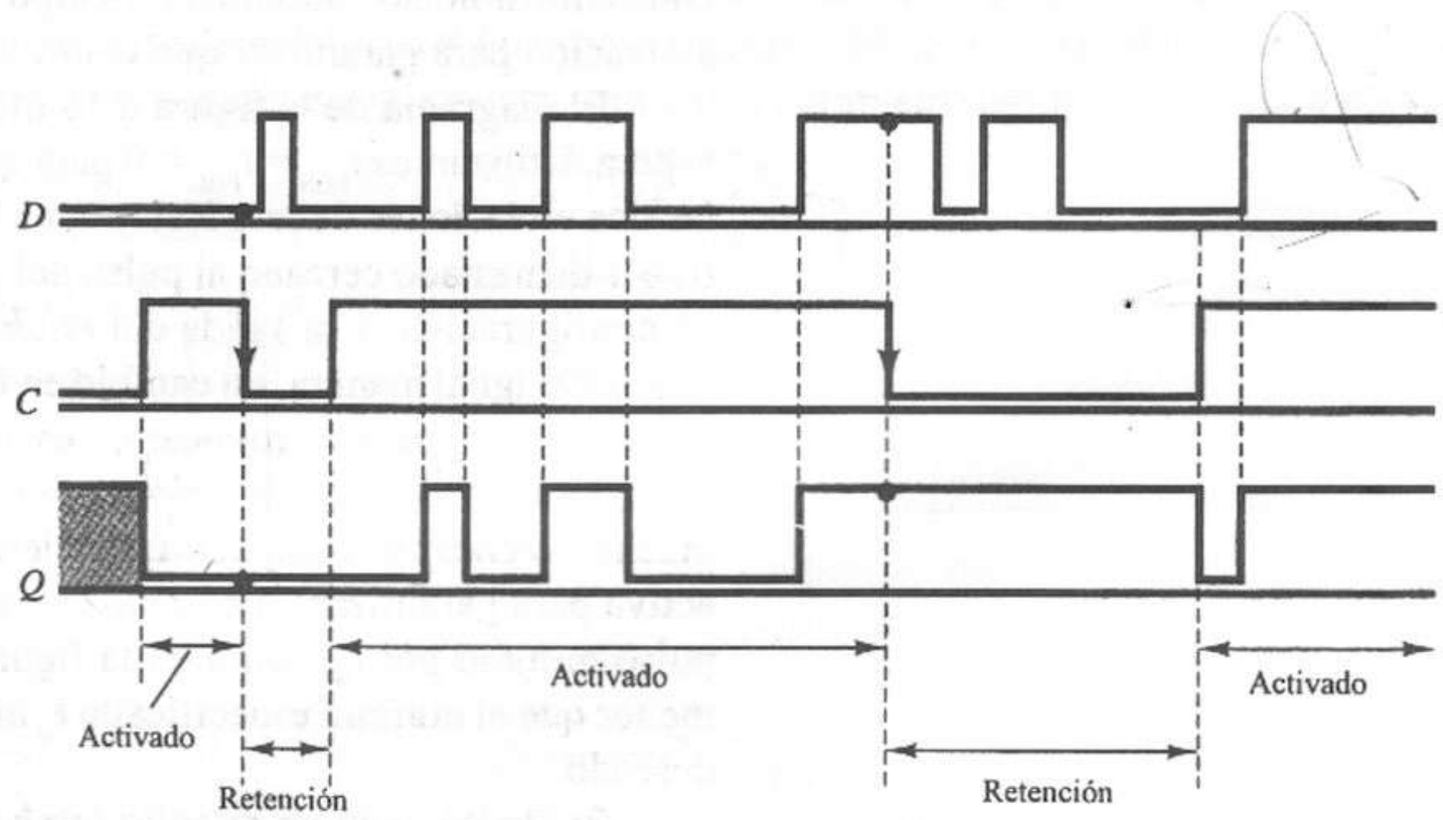


Figura 6.17 Diagrama de tiempos del *latch* D.

Tiempo de configuración, tiempo de retención y restricciones de ancho de pulso

Para garantizar que un valor específico en la entrada de excitación D determine el estado final del *latch*, D no debe poder cambiar demasiado cerca del momento

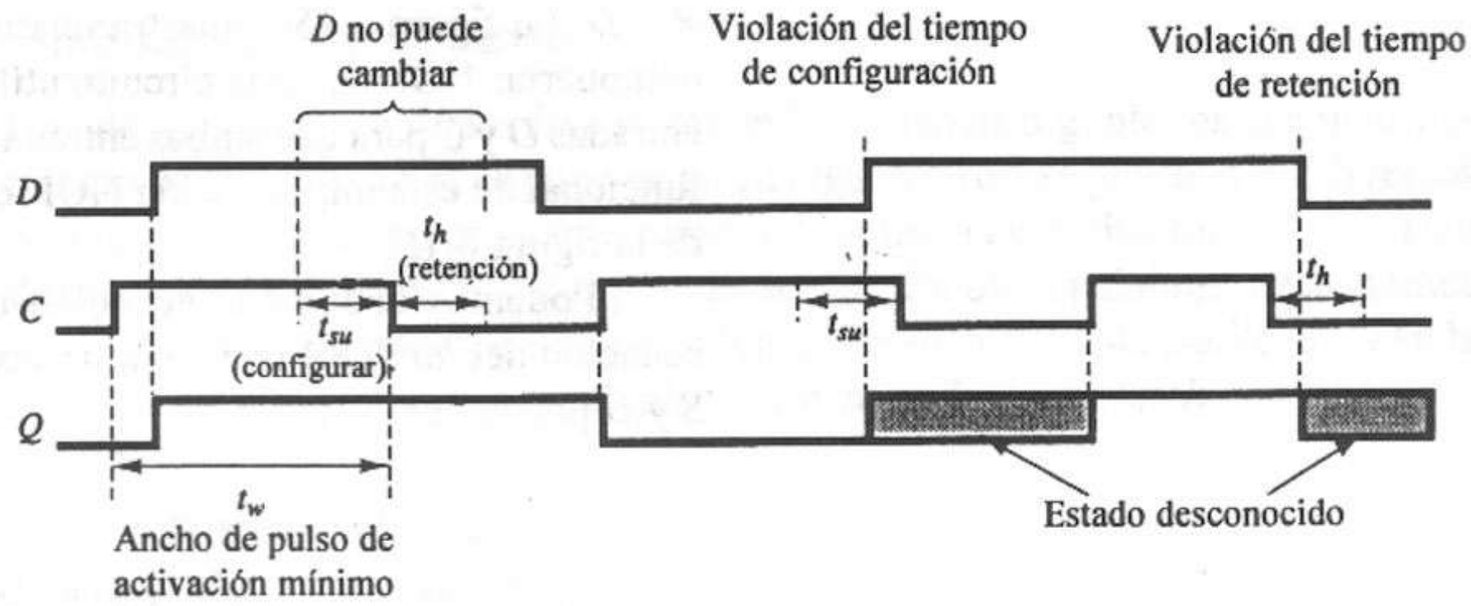


Figura 6.18 Restricciones de tiempos en el *latch* D

en que la señal de activación pasa de alta a baja. Debemos definir dos parámetros de restricción de tiempo para cada dispositivo *latch*, con el fin de identificar cuándo debe permanecer constante la entrada de excitación y garantizar su funcionamiento correcto.

Definimos el *tiempo de configuración* del *latch*, denotado por t_{su}, como el periodo de tiempo inmediato anterior a la transición de la señal de activación, durante el cual la entrada de excitación debe ser estable; es decir, la entrada de excitación debe estar "configurada" al menos un tiempo t_{su} antes de la transición de la señal de activación y no debe cambiar hasta después de terminar la transición.

Definimos el *tiempo de retención* del *latch*, denotado por t_h, como el periodo de tiempo inmediato posterior a la transición de señal de activación, durante el cual D no debe cambiar. Por tanto, la entrada de excitación debe permanecer constante al menos durante un tiempo t_h después de la transición de la señal de activación para garantizar que el *latch* mantendrá el valor correcto.

El diagrama de la figura 6.18 muestra los tiempos de configuración y de espera. Utilizamos $t_{PLH} = t_{PHL} = 0$ para que el diagrama sea más sencillo. Observe las dos violaciones de restricciones en el diagrama de tiempos. Un cambio en D $0 \rightarrow 1$ demasiado cercano al pulso del reloj representa una violación del tiempo de configuración, y la salida del *latch* podría o no cambiar de $0 \rightarrow 1$ como se desea. De igual manera, un cambio en D de $1 \rightarrow 0$ posterior pero muy cercano al pulso del reloj puede provocar un estado impredecible.

Además de las restricciones del tiempo de configuración y de retención, muchos *latches* con compuertas requieren un ancho de pulso mínimo en la entrada activa para garantizar un cambio correcto de estado. Denotamos este ancho de pulso mínimo por t_w, como en la figura 6.18. Cualquier pulso cuyo ancho sea menor que el mínimo especificado t_w no bastará para iniciar el cambio de estado deseado.

Por tanto, para un circuito *latch* con compuertas dado, el diseñador debe garantizar que todos los pulsos de activación tengan el ancho suficiente para provocar cambios de estado y que, para cualquier transición de entrada de activación en el instante T, no suceda un cambio de entrada de excitación dentro del periodo $[T - t_{su}, T + t_h]$.

Módulo *latch* D 74LS75 cuádruple

Muchos módulos TTL estándar tienen *latches* D. El módulo SN74LS75 tiene cuatro *latches D* biestables con entradas de activación. El circuito lógico para el

latch D utilizado en este dispositivo, que aparece en la figura 6.19a, es una implantación directa de la ecuación característica del *latch* D (ecuación 6.12).

Los modos de operación de este circuito son controlados por la señal de activación C. Cuando C tiene el valor 0 lógico, como en la figura 6.19b, la compuerta superior AND se desactiva, de modo que su salida envía un 0 lógico a la compuerta NOR. Una compuerta NOR con una de sus entradas en 0 lógico actúa como una compuerta NOT para su otra línea de entrada, como en la figura 6.19b. Observe también que al mantener C en 0 lógico se coloca un 1 lógico en la compuerta inferior AND, por lo que ésta se comporta como una compuerta transparente para la señal de retroalimentación del *latch*, Q. En este modo de operación, el dispositivo se asemeja a dos compuertas NOT configuradas como un elemento de almacenamiento biestable. Llamamos a éste el modo de *retención* o de *almacenamiento*.

Ahora suponga que cambiamos el valor de la entrada de activación C a un 1 lógico, como en la figura 6.19c. Este modo de operación activa una ruta desde la entrada D a través de la compuerta superior AND y la compuerta NOR hasta la salida del *latch*, Q ($Q = D$). En este modo, la entrada se dirige a través de las compuertas en forma directa hasta la salida, como se muestra en la figura 6.19c. Llamamos a éste el modo de operación *con compuertas*. Observe que el SN74LS75 ofrece los mismos modos de operación que el anterior diseño del *latch* D de la figura 6.15.

Para que el *latch* opere en forma correcta, debe existir un retardo entre los cambios en D o C y cualquier cambio correspondiente en la señal de retroalimentación Q, de modo que el siguiente estado, Q^*, quede determinado realmente por el estado presente, Q, y las entradas. Este retardo está dado por los retardos por propagación a través de las compuertas del circuito *latch* de dos niveles. Con frecuencia, mostramos este retardo como en el modelo de circuito secuencial para el *latch* de la figura 6.19d, donde modelamos los retardos de las compuertas lógicas como un único parámetro agrupado, Δt. La tabla 6.2

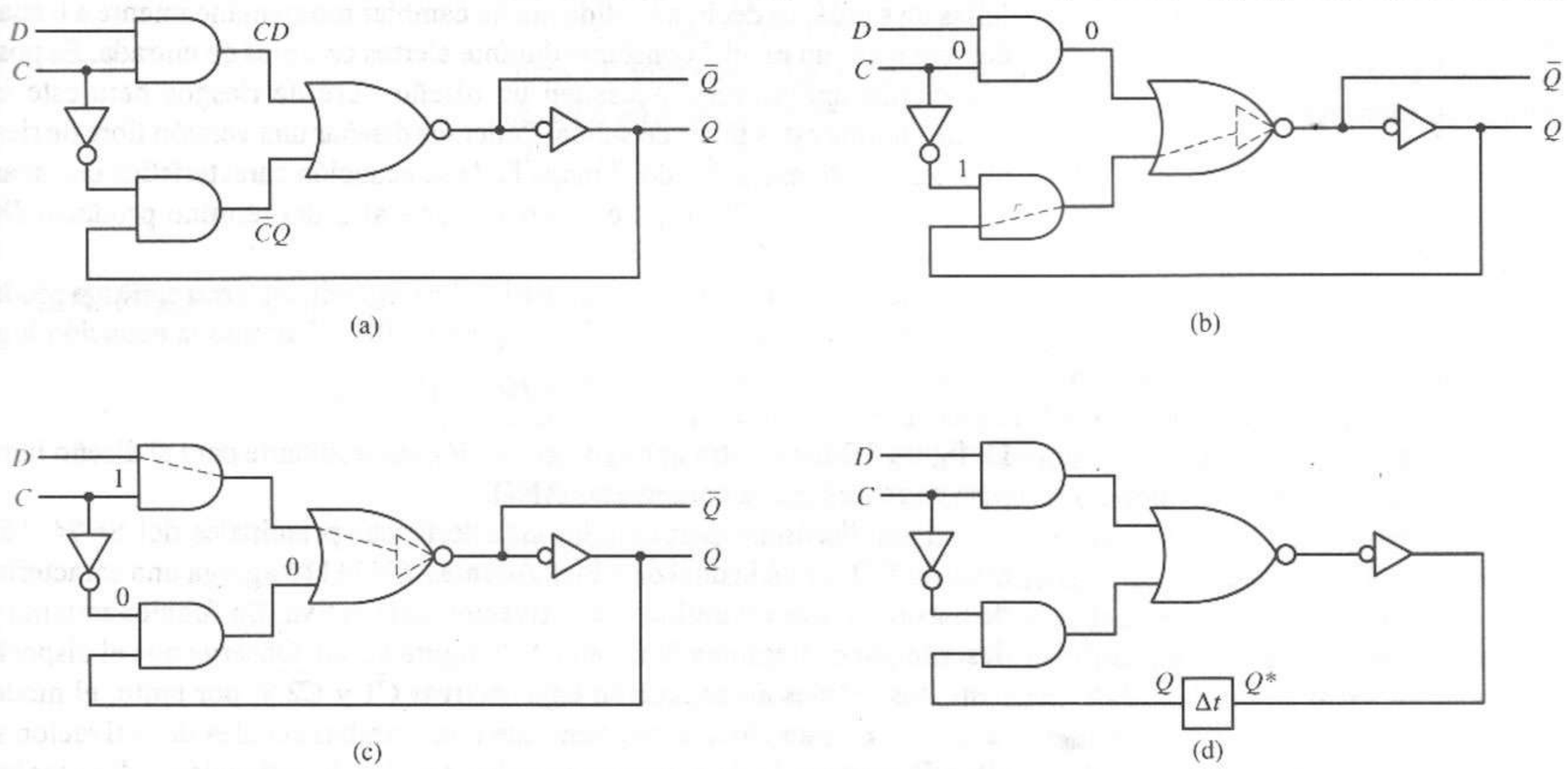


Figura 6.19 Una segunda configuración de *latch* D, 74LS75. **(a)** Diagrama lógico. **(b)** Modo de retención o almacenamiento. **(c)** Modo de compuerta. **(d)** Modelo de circuito secuencial. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

enumera los tiempos de retardo y los valores de las restricciones de tiempo típicos para 74LS75.

TABLA 6.2 RETARDOS POR PROPAGACIÓN Y RESTRICCIONES DE TIEMPOS PARA EL SN74LS75 [1]

Parámetros de tiempos	Valores típicos (ns)
t_{PLH} $(D \rightarrow Q)$	15
t_{PHL} $(D \rightarrow Q)$	9
t_{PLH} $(D \rightarrow \bar{Q})$	12
t_{PHL} $(D \rightarrow \bar{Q})$	7
t_{PLH} $(C \rightarrow Q)$	15
t_{PHL} $(C \rightarrow Q)$	14
t_{PLH} $(C \rightarrow \bar{Q})$	16
t_{PHL} $(C \rightarrow \bar{Q})$	7
t_{su} (tiempo de configuración mínimo para D)	20
t_h (tiempo de retención mínimo para D)	5
t_w (ancho de pulso mínimo en C)	20

Módulo *latch* D 74116 dual de 4 bits libre de riesgos

El circuito *latch* D de la figura 6.19a utiliza un número mínimo de compuertas lógicas. Sin embargo, contiene un riesgo estático 1, lo que lo hace propenso a fallas de salida; es decir, su salida puede cambiar momentáneamente a 0 cuando debiera tener un nivel 1 constante durante ciertos cambios de entrada. Es posible que ciertas aplicaciones necesiten un diseño libre de riesgos para este *latch* D que elimine estas fallas de salida. Podemos diseñar una versión libre de riesgos de este circuito examinando el mapa K de su ecuación característica que aparece en la figura 6.20a. El riesgo estático 1 ocurre al ir del término producto DC al término producto $Q\overline{C}$ en este mapa.

Podemos eliminar el riesgo estático 1 agregando un tercer término producto DQ, como muestra el mapa K de la figura 6.20b, formando la ecuación lógica

$$Q^* = DC + \bar{C}Q + DC$$

La figura 6.20c muestra que el diagrama lógico resultante para el diseño libre de riesgos requiere tres compuertas AND.

Este diseño proporciona las características principales del SN74116, un módulo TTL *latch* D dual de 4 bits. Además, SN74116 agrega una característica de borrado (clear) y utiliza una activación baja activa. Su funcionamiento está descrito por el diagrama funcional de la figura 6.20d. Observe que el dispositivo tiene dos señales de activación bajas activas $\overline{C1}$ y $\overline{C2}$ y, por tanto, el modo de retención de este circuito requiere que una o ambas señales de activación sean altas. Para el modo de compuerta, ambas señales de activación deben ser bajas.

Observe también que se proporciona una señal de *reset*, $\overline{CLR}$, cambiando la compuerta NOT de salida por una compuerta NOR. Un 0 lógico en la línea

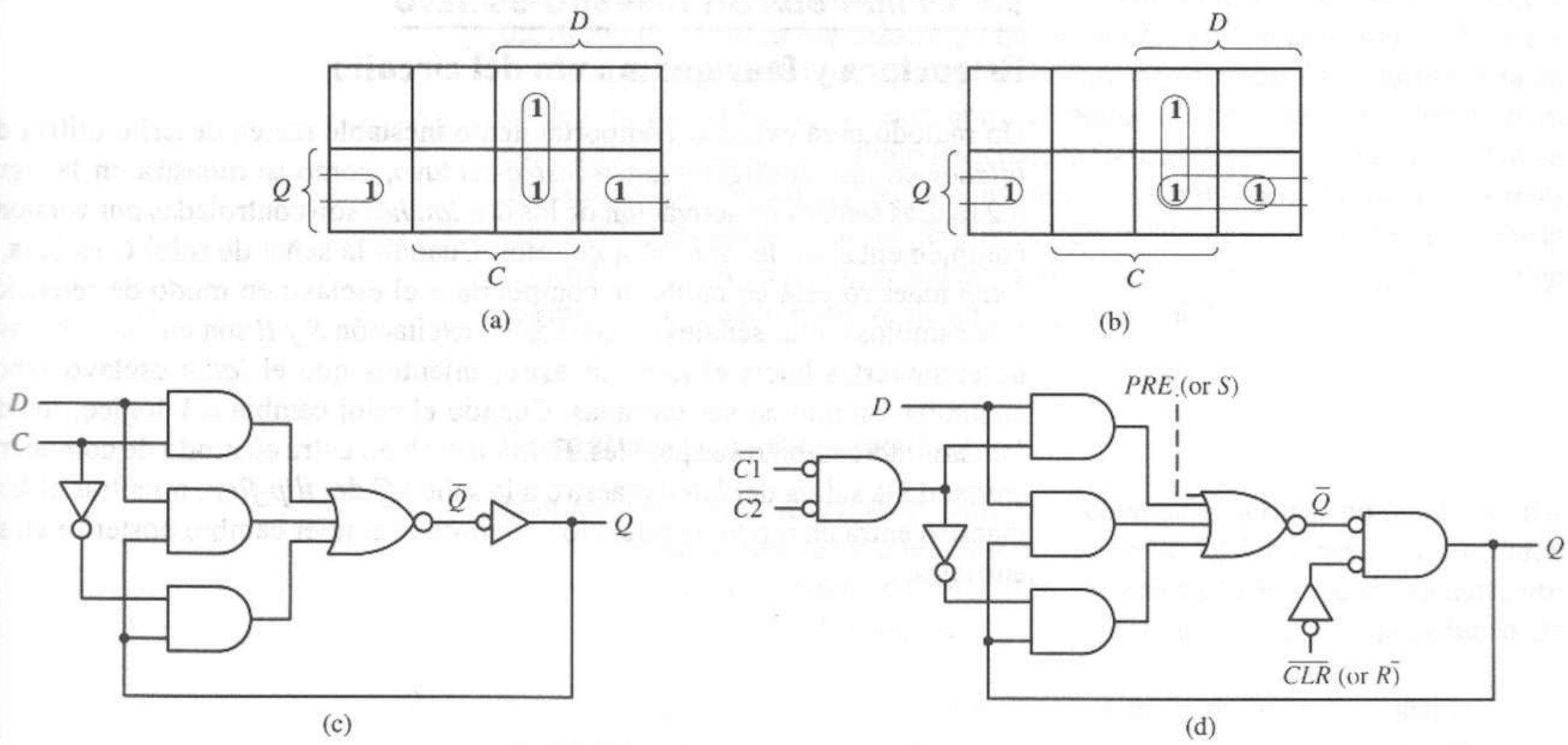


Figura 6.20 *Latch* D libre de riesgos (SN74116). **(a)** Mapa K para Q^* de un *latch* D. **(b)** Mapa K para un *latch* D libre de riesgos. **(c)** Diagrama lógico de un *latch* D libre de riesgos. **(d)** Diagrama funcional de SN74116.

$\overline{CLR}$ hace que la salida Q asuma el valor 0 lógico. Además, podríamos agregar una entrada de control *PRE* (de *preset*) para que la salida Q asuma el valor 1 lógico, como muestra la línea punteada, si estuviera disponible un *pin* adicional. Las entradas de control $\overline{CLR}$ y *PRE* de la figura 6.20d son líneas de control *asíncronas*, pues afectan el estado del *latch* en forma directa, sin estar sincronizadas con la señal de activación. Con frecuencia, grupos de *latches* se llevan a un estado inicial deseado enviando un pulso a sus entradas de control $\overline{CLR}$ o *PRE* mediante una única línea de señal *RESET*.

6.4 *Flip-flops*

Los circuitos de *latch* presentados hasta el momento no son adecuados para usarse en circuitos lógicos secuenciales síncronos. Cuando la señal de activación C está activa, las entradas de excitación se envían directamente mediante compuertas a la salida Q. Así, cualquier cambio en la entrada de excitación provoca de inmediato un cambio en la salida del *latch*. Recordemos nuestro modelo para el circuito secuencial síncrono de la figura 6.1. Las señales de salida de los elementos de memoria son las señales de entrada a la lógica combinatoria y viceversa. Cuando su habilitación está activa, ¡un *latch* actúa también como un circuito combinatorio! Así, podríamos tener dos circuitos combinatorios en cascada alimentándose entre sí, generando oscilaciones y un comportamiento transitorio inestable. Resolvemos este problema con una señal de control de tiempo especial llamada *reloj*, la cual restringe los instantes en que podemos cambiar los estados de los elementos de memoria.

6.4.1 *Flip-flops* SR maestro-esclavo

Estructura y funcionamiento del circuito

Un método para evitar el comportamiento inestable recién descrito utiliza dos *latches* en una configuración *maestro-esclavo*, como se muestra en la figura 6.21a. Las señales de activación de los dos *latches* son controladas por versiones complementarias de una señal de reloj. Cuando la señal de reloj C es baja, el *latch* maestro está en modo de compuerta y el esclavo en modo de retención. Los cambios en las señales de entrada de excitación S y R son enviadas a través de compuertas hacia el *latch* maestro, mientras que el *latch* esclavo ignora cualquier cambio en sus entradas. Cuando el reloj cambia a 1 lógico, los dos *latches* intercambian sus papeles. El *latch* esclavo entra en modo de compuerta, enviando la salida del *latch* maestro a la salida Q del *flip-flop*, mientras el *latch* maestro entra en modo de retención e ignora cualquier cambio posterior en sus entradas.

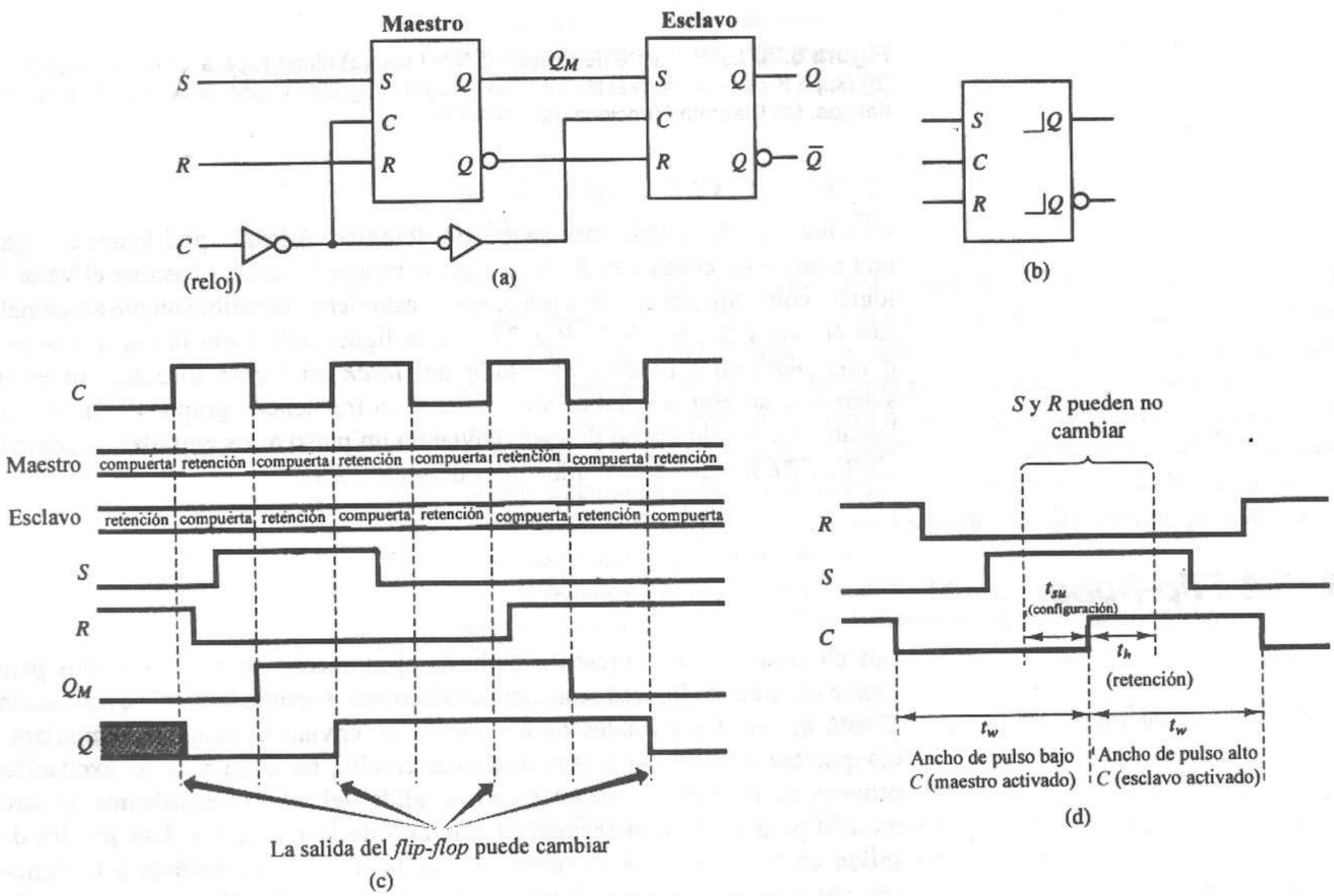


Figura 6.21 *Flip-flop* SR maestro-esclavo. **(a)** Diagrama lógico. **(b)** Símbolo lógico del dispositivo disparado por pulsos. **(c)** Comportamiento respecto del tiempo. **(d)** Restricciones de tiempos.

A veces, se dice que los *flip-flops* maestro-esclavo como el de la figura 6.21a son *disparados por pulsos,* ya que requieren ambas transiciones $0 \rightarrow 1$ y $1 \rightarrow 0$ en la entrada de reloj para operar en forma adecuada. El maestro opera en una de las transiciones; es decir, entra al modo de activación; el esclavo opera en la otra transición. El símbolo lógico de la figura 6.21b indica la naturaleza de disparo por pulsos del dispositivo, mostrando la transición del pulso del reloj que habilita al esclavo en las terminales de salida Q y $\overline{Q}$ del *flip-flop*. En la figura 6.21b, la transición ascendente indica que las salidas Q y $\overline{Q}$ del *flip-flop* cambian en el flanco positivo de un pulso de la señal de reloj.

Características de tiempos

Si el *flip-flop* SR se utiliza en un circuito secuencial síncrono, no puede ocurrir una oscilación inestable ya que, en todo momento, el *latch* maestro o el *latch* esclavo están en el modo de retención, bloqueando todo comportamiento transitorio inestable. La figura 6.21c muestra este comportamiento de tiempos.

Observe que las entradas S y R del *latch* maestro deben ser estables antes de la transición de reloj que lleva al maestro al modo de retención. Por tanto, las entradas del *flip-flop* están sujetas a las mismas restricciones de tiempo de configuración y espera descritas con anterioridad para los *latches* con compuertas. La figura 6.21d muestra los tiempos de configuración y espera para el *flip-flop* SR de la figura 6.21a. Como las entradas de excitación sólo afectan al *latch* maestro, definimos los tiempos de configuración y de retención relativos a la transición ascendente de la señal de reloj, que es la transición del reloj que cambia el modo del *latch* maestro, del modo de compuertas al modo de retención. Las entradas de excitación del *latch* esclavo están conectadas con las salidas del *latch* maestro y, por tanto, no son afectadas en forma directa por las entradas de excitación externas.

La figura 6.21d ilustra también las restricciones de ancho de pulso mínimo del reloj para el *flip-flop* maestro-esclavo. El parámetro mínimo para el ancho de pulso es el ancho mínimo necesario para el funcionamiento adecuado del *latch* maestro, mientras que el parámetro máximo para el ancho de pulso es el ancho mínimo necesario para el *latch* esclavo. La suma de estos dos anchos de pulso determina el periodo mínimo de cualquier señal de reloj que será utilizada por el *flip-flop*.

Tabla de excitación y ecuación característica

Las figuras 6.22a y b muestran la tabla de excitación y el diagrama de estados, respectivamente, para el *flip-flop* maestro-esclavo SR. Observe que las columnas S, R y Q de la tabla de excitación denotan las condiciones en las señales del *flip-flop antes* de aplicar el pulso del reloj. La columna Q^* denota la salida del *flip-flop después* de aplicar el pulso del reloj. Al comparar esta tabla con la figura 6.11a, vemos que el funcionamiento del *flip-flop* SR maestro-esclavo es similar al del *latch* SR simple. De igual manera, los diagramas de estados son idénticos, aunque el *latch* cambia de estado en forma inmediata cuando S y R cambian, mientras que todos los cambios de estado del *flip-flop* son activados

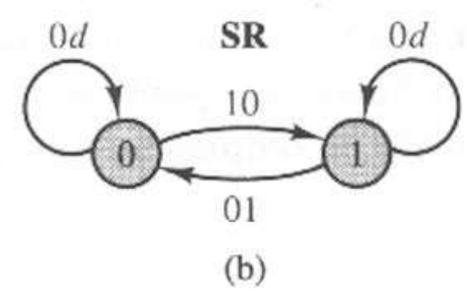
por los pulsos del reloj. En consecuencia, una misma ecuación característica describe el funcionamiento de ambos dispositivos:

$$Q^* = S + \bar{R}Q \tag{6.13}$$

La diferencia es que la salida del *latch* reacciona de inmediato a cualquier cambio en la entrada, mientras que los cambios en la salida del *flip-flop* están controlados por el pulso de reloj C. Observe que C debe tener transiciones negativa y positiva.

S	R	Q	C	Q^*	
0	0	0	⎍	0	Sin cambio
0	0	1	⎍	1	
0	1	0	⎍	0	Reset
0	1	1	⎍	0	
1	0	0	⎍	1	Set
1	0	1	⎍	1	
1	1	0	⎍	×	No permitido
1	1	1	⎍	×	

(a)



(b)

Figura 6.22 Características del *flip-flop* SR maestro-esclavo. **(a)** Tabla de excitación. **(b)** Diagrama de estados.

6.4.2 *Flip-flops* D maestro-esclavo

Podemos construir un *flip-flop* D maestro-esclavo con dos *latches* D, como en la figura 6.23a. Observe que este *flip-flop* opera de la misma manera que la versión SR de la figura 6.22. El *latch* maestro se liga con las compuertas cuando el reloj es bajo y el *latch* esclavo lo hace cuando el reloj es alto. La figura 6.23b muestra el símbolo lógico para este dispositivo disparado por pulsos. Observe que el símbolo lógico indica que las salidas cambian en la transición positiva de un pulso de la señal de reloj.

Las figuras 6.24a y 6.24b muestran la tabla de excitación y el diagrama de estados, respectivamente, del *flip-flop* D maestro-esclavo. El diagrama de tiempos de la figura 6.24c ilustra el comportamiento de este dispositivo. En la parte superior del diagrama, indicamos el *latch* con compuertas mediante los símbolos M y S, de maestro y esclavo. Cuando $C = 0$, el maestro está conectado, de modo que su entrada pasa al esclavo. En la transición $0 \rightarrow 1$ de C, el maestro "captura" el valor de entrada en D (denotado mediante el símbolo ×) y conserva este valor. Como el esclavo se conecta si $C = 1$, el valor capturado en el maestro pasa a la salida Q del *flip-flop*. En la transición descendente del reloj C, el esclavo "captura" los datos del maestro, como muestran los símbolos × sobre la señal Q_M del diagrama. Observe que el diagrama de tiempos incluye los retardos t_{PLH} y t_{PHL}.

Podemos resumir el comportamiento general de la salida Q del *flip-flop* D observando que Q asumirá el valor de D en la transición ascendente del reloj C. Por tanto, la ecuación característica para un *flip-flop* D maestro-esclavo es simplemente

$$Q^* = D \tag{6.14}$$

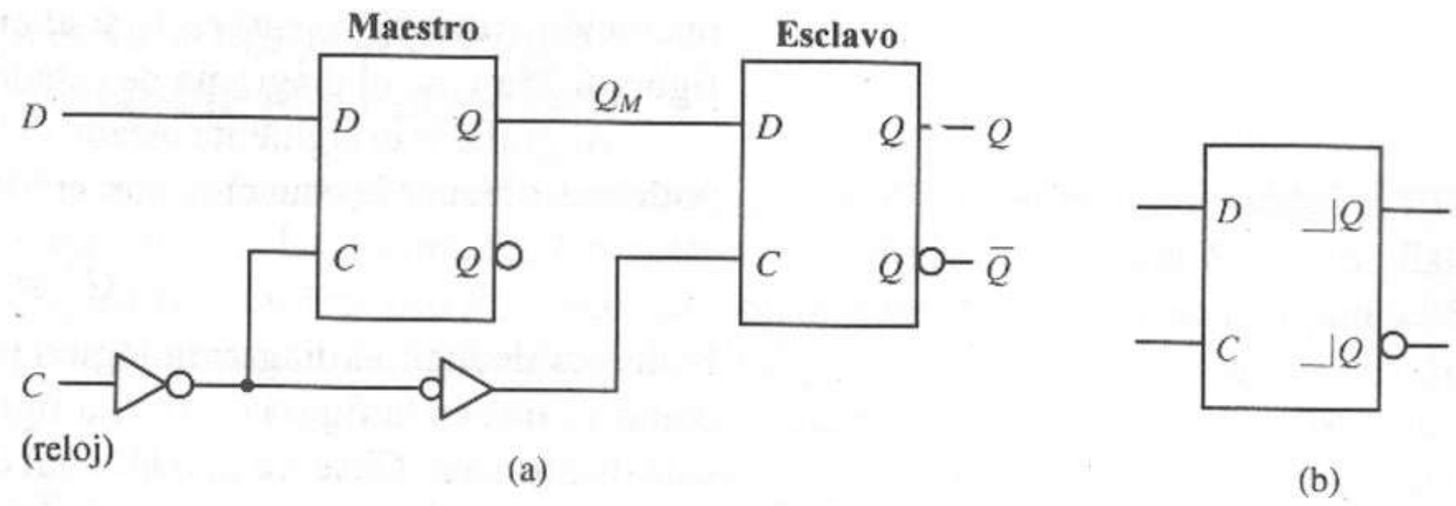


Figura 6.23 *Flip-flop* D maestro-esclavo. **(a)** Diagrama lógico. **(b)** Símbolo lógico.

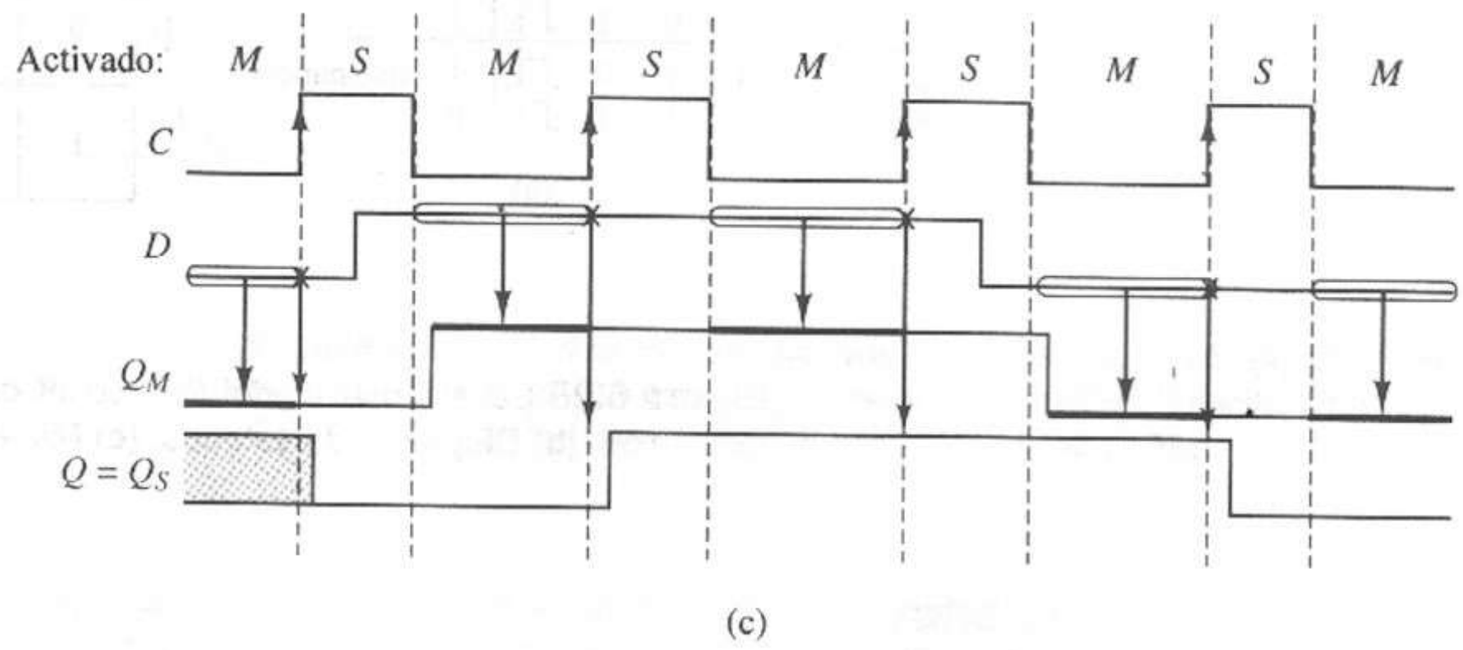


Figura 6.24 Características del *flip-flop* D maestro-esclavo. **(a)** Tabla de excitación. **(b)** Diagrama de estados. **(c)** Diagrama de tiempos.

6.4.3 *Flip-flops* JK maestro-esclavo

Estructura y funcionamiento del circuito

Podemos considerar el *flip-flop* JK como una extensión del diseño SR ya analizado. El JK opera como un *flip-flop* SR a cuyas entradas se asigna $J = S$ y $K = R$. Sin embargo, en tanto que la combinación $S = R = 1$ no está permitida, el JK utiliza este caso particular para agregar un modo de operación muy útil. La característica adicional del dispositivo JK es que su estado se *alterna*; es decir, cambia del $0 \rightarrow 1$ o $1 \rightarrow 0$ cuando $J = K = 1$. Resumimos los cuatro modos de

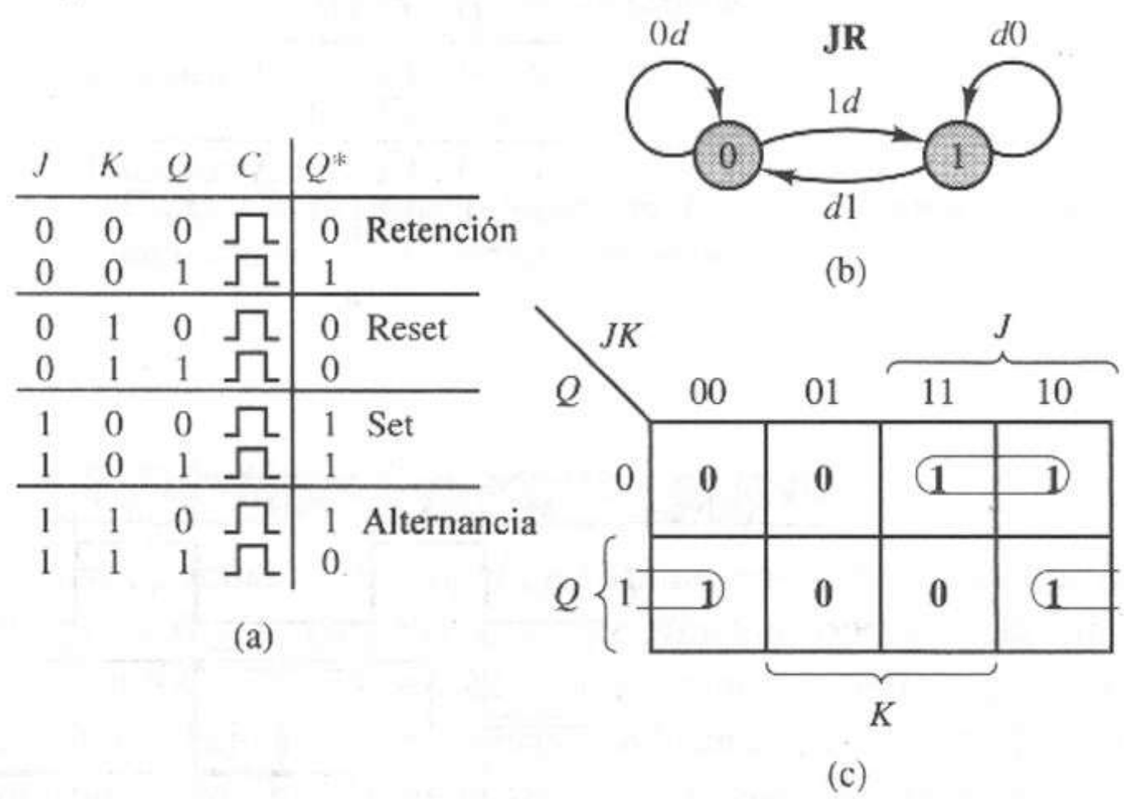
operación (retención, *set*, *reset* y alternancia) en la tabla de excitación de la figura 6.25a y en el diagrama de estados correspondiente de la figura 6.25b.

Al graficar el siguiente estado Q^* en un mapa K, como en la figura 6.25c, podemos obtener la ecuación característica del *flip-flop* JK:

$$Q^* = \bar{K}Q + J\bar{Q} \tag{6.15}$$

Podemos deducir el diagrama lógico para el *flip-flop* a partir de esta ecuación, como vemos en la figura 6.26a. La figura 6.26b muestra el símbolo lógico para este dispositivo. Observe que la señal de entrada del reloj se invierte dentro del propio dispositivo de modo que el esclavo cambie en la transición descendente del reloj.

Examine el diagrama de estados de la figura 6.25b. El *flip-flop* JK pasará del estado 0 al estado 1 con una entrada $J = 1$ y $K = 0$ (*set*) o $J = 1$ y $K = 1$ (alternancia). Es decir, un 1 lógico en J lleva el dispositivo al estado 1, sin importar el valor de la entrada K. Por tanto, K es una condición prescindible, lo que denotamos en el diagrama de estados con el valor d. El resto del diagrama se puede deducir de la tabla de excitación.



J	K	Q	C	Q^*	
0	0	0	⎍	0	Retención
0	0	1	⎍	1	
0	1	0	⎍	0	Reset
0	1	1	⎍	0	
1	0	0	⎍	1	Set
1	0	1	⎍	1	
1	1	0	⎍	1	Alternancia
1	1	1	⎍	0	

(a)

Figura 6.25 Características del *flip-flop* JK disparado por pulsos. **(a)** Tabla de excitación. **(b)** Diagrama de estados. **(c)** Mapa K para Q^*.

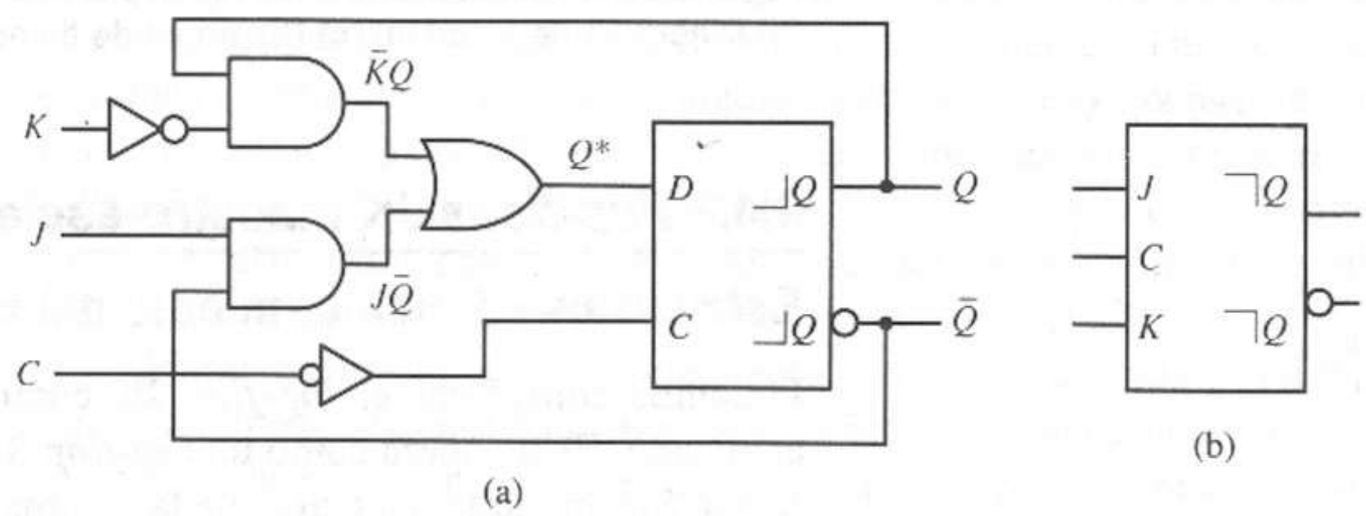


Figura 6.26 *Flip-flop* JK disparado por pulsos. **(a)** Diagrama lógico. **(b)** Símbolo lógico.

Módulo *flip-flop* JK dual disparado por pulsos 7476

Hay varios *flip-flops* JK disparados por pulsos disponibles como módulos TTL estándar [1]. La figura 6.27 muestra el símbolo lógico del SN7476. Este dispositivo contiene dos *flip-flops* que operan como se muestra en la figura 6.26. La configuración incluye señales *set* $\overline{PRE}$ y *reset* $\overline{CLR}$ asíncronas. Las señales $\overline{PRE}$ y $\overline{CLR}$ tienen precedencia sobre el funcionamiento de las entradas disparadas por pulsos J, K y CLK; es decir, si $\overline{CLR} = 0$, el estado Q^* asume el valor 0, o si $\overline{PRE} = 0$, el estado Q^* asume el valor 1, con independencia de los valores del reloj y de las entradas de excitación.

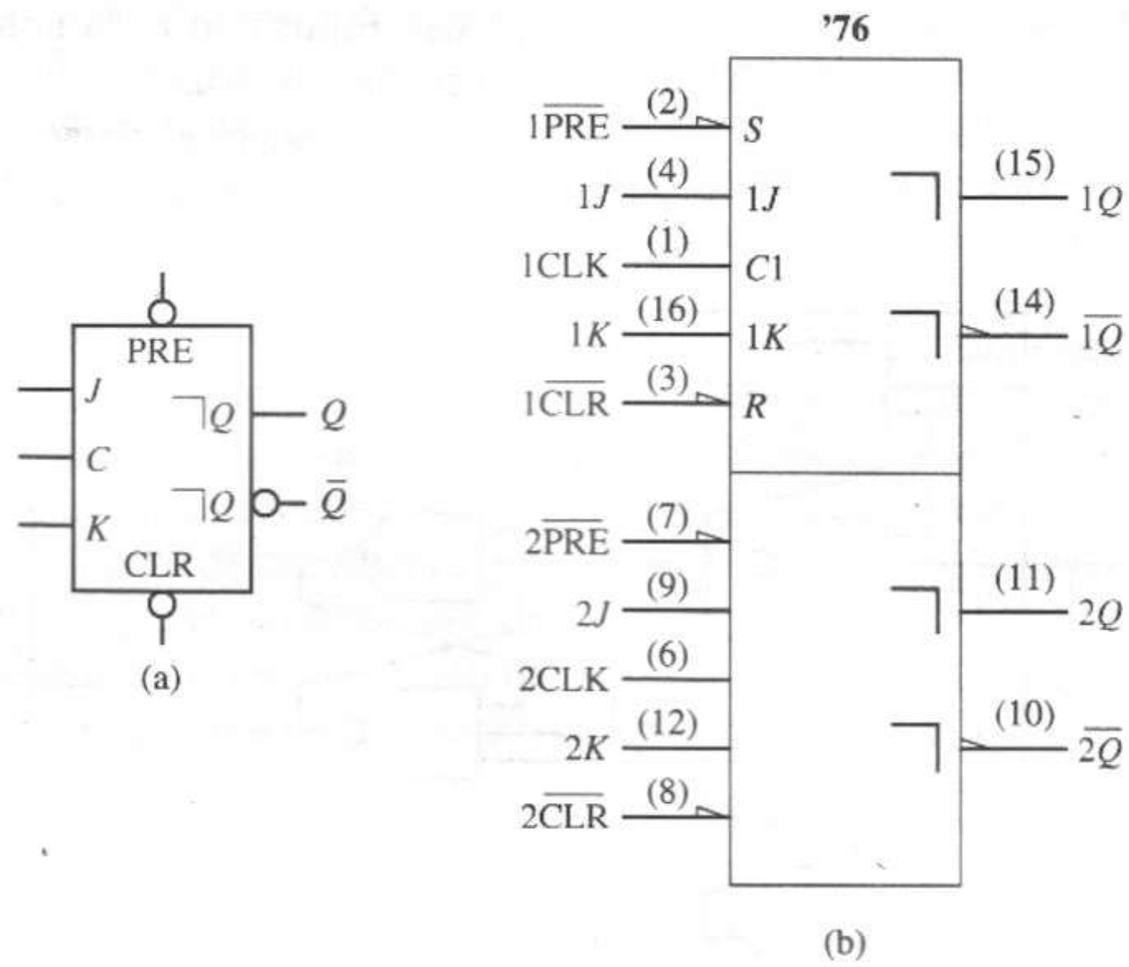


Figura 6.27 *Flip-flop* JK dual disparado por pulsos, 7476. **(a)** Símbolo lógico genérico. **(b)** Símbolo lógico estándar de IEEE. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

6.4.4 *Flip-flops* D disparados por transición

Todos los *flip-flops* disparados por pulsos descritos en la sección 6.4.3 necesitan una transición ascendente del reloj y una descendente para su funcionamiento adecuado. La disposición maestro-esclavo que hemos presentado tenía un mecanismo de almacenamiento temporal para eliminar las condiciones transitorias inestables en los circuitos secuenciales con elementos de retroalimentación. Otro método para resolver el problema de los estados transitorios inestables es diseñar los circuitos del *flip-flop* de modo que sea sensible a sus entradas de excitación sólo durante las transiciones ascendentes o descendentes del reloj. Se dice que un circuito con esta característica de diseño *se dispara por transición positiva* si responde a una transición de reloj $0 \rightarrow 1$, o que *se dispara por transición negativa* si responde a una transición de reloj $1 \rightarrow 0$. Esta característica de sensibilidad a las transiciones elimina los estados transitorios inestables al reducir de manera drástica el periodo de aplicación de las señales de excitación de entrada a los *latches* internos.

Los módulos *flip-flop* D comerciales tienen por lo general una señal de control disparada por transición positiva. A continuación describiremos algunos de estos dispositivos, enumerados en la tabla 6.1.

Módulo *flip-flop* D 7474 dual disparado por transición positiva

Consideremos el diagrama lógico del *flip-flop* D SN7474 dual disparado por transición positiva de la figura 6.28a. Este circuito examina la señal de entrada de excitación *D* durante la transición ascendente de la entrada de reloj *CLK*. Los símbolos genérico y estándar IEEE del SN7474 aparecen en las figuras 6.28b y c, respectivamente. Es importante observar que el pequeño triángulo en la entrada *C*1 del dispositivo es la notación estándar para indicar que es disparado por transición positiva.

Los modos de operación del SN7474 aparecen en la tabla de excitación de la figura 6.29. Observe que las señales *preset* y *clear* asíncronas, $\overline{CLR}$ y $\overline{PRE}$,

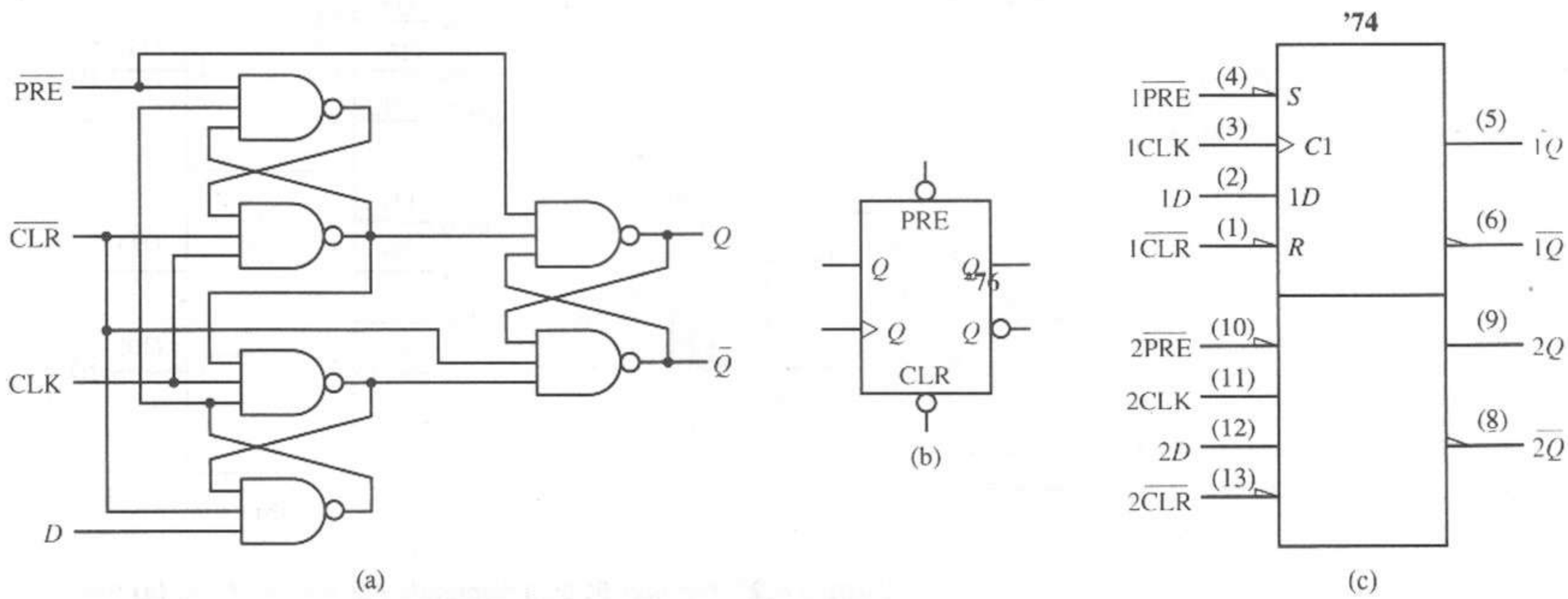


Figura 6.28 *Flip-flop* D dual disparado por transición positiva, SN7474. **(a)** Diagrama lógico. **(b)** Símbolo lógico genérico. **(c)** Símbolo lógico estándar de IEEE. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

Entradas				Salidas		
$\overline{PRE}$	$\overline{CLR}$	D	CLK	Q	$\overline{Q}$	Modo
L	H	×	×	H	L	*Set*
H	L	×	×	L	H	*Clear*
L	L	×	×	H	H	No permitido
H	H	H	↑	H	L	Funcionamiento con reloj
H	H	L	↑	L	H	Funcionamiento con reloj
H	H	×	L	Q_0	$\overline{Q}_0$	Retención

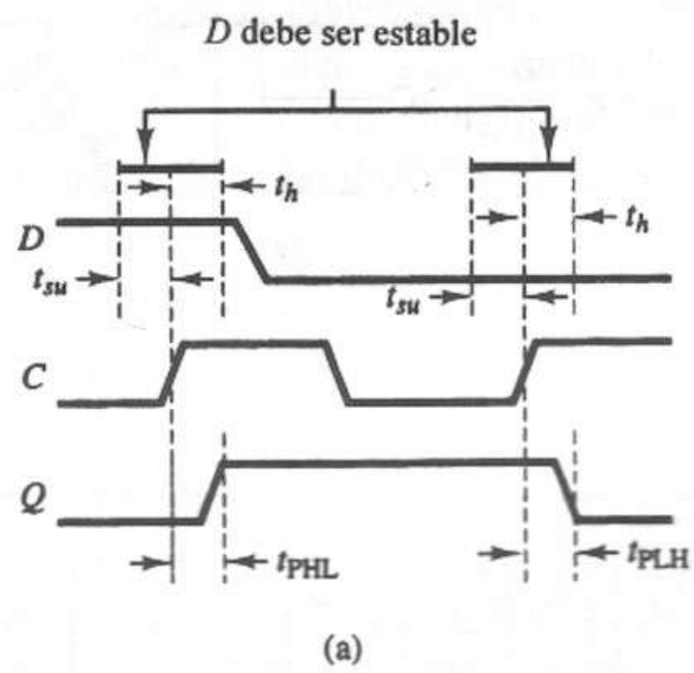
Figura 6.29 Tabla de excitación del SN7474. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

tienen precedencia sobre el funcionamiento del circuito controlada por el reloj. Cuando $\overline{CLR}$ y $\overline{PRE}$ están inactivos (altos), el reloj *CLK* controla el dispositivo. Cuando *CLK* está bajo, el *flip-flop* está en el modo de retención. Sin embargo, cuando hay una transición $0 \rightarrow 1$ del reloj, denotada por $\uparrow$, la entrada de datos *D* se transfiere a la salida *Q* del *flip-flop*.

Características temporales del *flip-flop* disparado por transición

Para garantizar el funcionamiento adecuado de cualquier *flip-flop* disparado por transición, las entradas de excitación no deben cambiar inmediatamente antes o después de la transición del reloj. Los límites precisos de estos periodos de tiempo para cada tipo de *flip-flop* están especificados en el manual TTL [1]. Como habíamos definido para los *latches* y los *flip-flops* disparados por pulsos, el periodo anterior a la transición de reloj para un *flip-flop* disparado por transición es el *tiempo de configuración* (t_{su}); el periodo posterior a la transición es el *tiempo de retención* (t_h). En general, si violamos las restricciones específicas de un dispositivo TTL disparado por transición, no queda garantizado el comportamiento del dispositivo. La figura 6.30 muestra las relaciones entre estas especificaciones de tiempos y los tiempos de retardo por propagación del *flip-flop* y la transición de reloj para un *flip-flop* D genérico disparado por transición positiva. Observe que los retardos por propagación desde el instante en que el reloj cruza el umbral de la transición ascendente hasta el cambio de la salida *Q* se llaman t_{PHL} y t_{PLH}, como definimos antes.

Examinemos el caso específico del SN7474. La figura 6.30b muestra los valores de t_{PHL} y t_{PLH} para este dispositivo, especificados en el manual TTL [1] como 0 ns.



A la salida *Q* desde:	Parámetro de retardo	Valor (ns)
Reloj	t_{PLH}	25
	t_{PHL}	40
PRE	t_{PLH}	25
	t_{PHL}	40
CLR	t_{PLH}	25
	t_{PHL}	40

(b)

Pin de entrada	Restricción	Valor mínimo (ns)
D	t_{su}	20
D	t_h	5
Reloj	t_w baja	30
Reloj	t_w alta	37
CLR	t_w baja	30
PRE	t_w alta	30

(c)

Figura 6.30 Especificaciones de tiempos del *flip-flop* SN7474 [1]. **(a)** Diagrama de tiempos. **(b)** Retardos por propagación. **(c)** Restricciones del tiempo. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

En otras palabras, para el SN7474, el valor de *D* se muestrea y transfiere a la salida *Q* del *flip-flop* en el preciso instante en que el reloj alcanza su valor de umbral. Debemos asegurar que la entrada tenga el valor 1 o 0 lógico en este instante, de modo que la salida *Q* del *flip-flop* sea el valor planeado en el diseño del sistema. La figura 6.30c muestra las restricciones de tiempos para el SN7474.

Módulos *flip-flop* D 74175 y 74273 disparados por transición positiva

La figura 6.31 muestra otros dos miembros de la familia TTL de los *flip-flops* D disparados por transición positiva. El *flip-flop* D cuádruple SN74175, que aparece

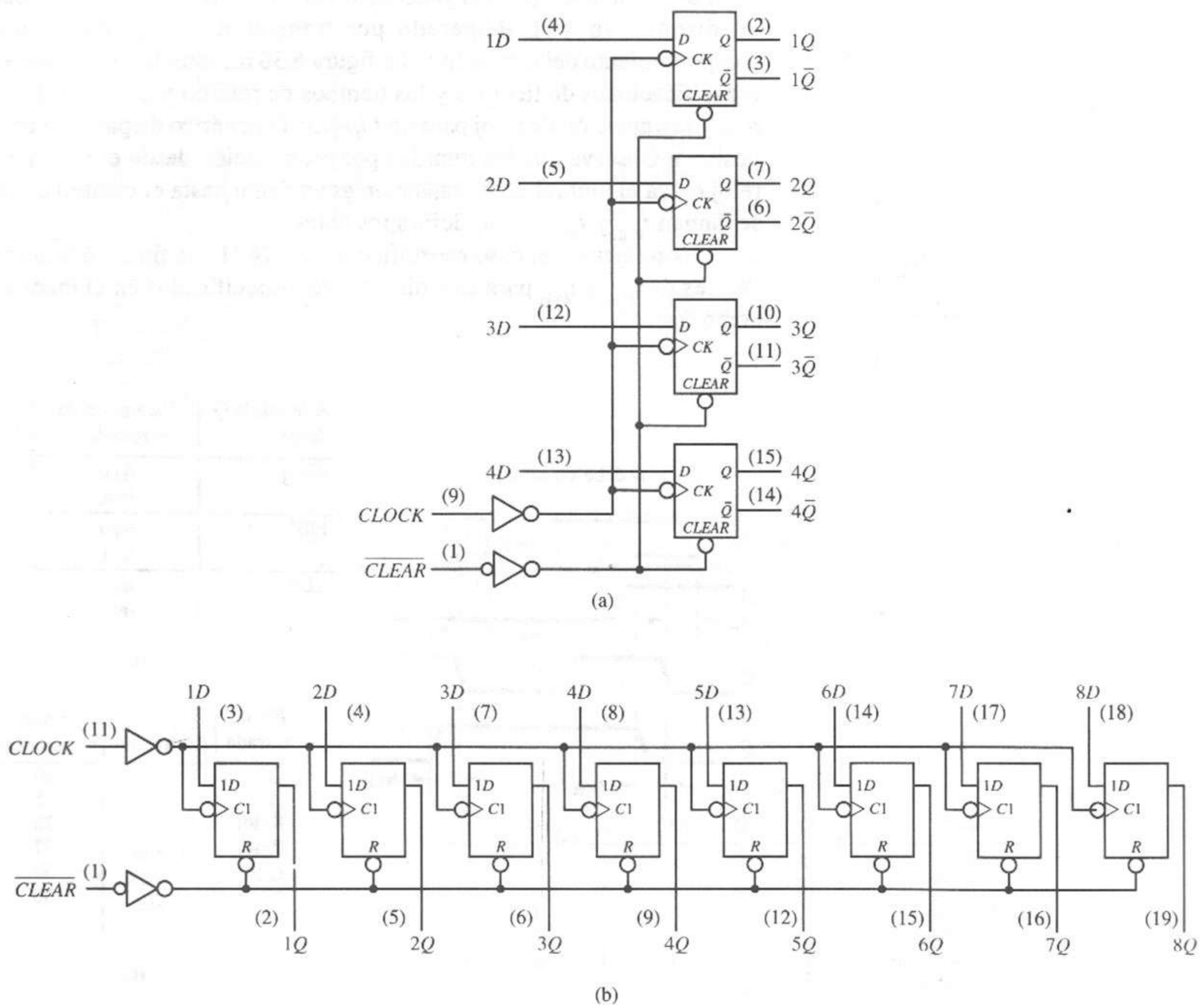


Figura 6.31 Paquetes de *flip-flop* D disparados por transición positiva. **(a)** SN74175. **(b)** SN74273. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

en la figura 6.31a, tiene controles de reloj y *clear* comunes, así como salidas real (Q) y complementada ($\overline{Q}$). El *flip-flop* D octal SN74273, que aparece en la figura 6.31b, tiene las mismas líneas de reloj y *clear* comunes, pero sólo saca las salidas reales (Q) a través de los *pines* del paquete. En los diagramas lógicos de estos dos dispositivos, observe los símbolos lógicos utilizados para los *flip-flops* D. La entrada de reloj *CK* lleva el pequeño triángulo que significa que el *flip-flop* se dispara por transición. La burbuja de inversión frente al triángulo indica un dispositivo disparado por transición negativa. Pero como la señal de entrada *CLOCK* está invertida por la compuerta NOT en la parte inferior del diagrama lógico, desde el punto de vista de los *pines* externos del paquete, los *flip-flops* parecen estar disparados por transición positiva.

El SN74273 tiene requisitos de tiempo de configuración y de espera de 20 y 5 ns, respectivamente. Estos valores están dentro de las tolerancias necesarias para evitar los estados transitorios inestables en la mayor parte de los diseños de circuitos lógicos secuenciales síncronos.

6.4.5 *Flip-flops* JK disparados por transición

Los *flip-flops* JK disparados por transición son comunes en la familia TTL. La mayor parte de ellos son disparados por transición negativa. Consideremos los siguientes ejemplos.

Flip-flop JK 74LS73A dual disparado por transición negativa

Examinemos el diagrama lógico de un SN74LS73A de la figura 6.32a. Este dispositivo dual disparado por transición negativa requiere tiempos de configuración y de espera de 20 y 0 ns, respectivamente. Las figuras 6.32b y c muestran sus símbolos lógicos genérico y estándar IEEE. Observe que este dispositivo de 14 *pines* tiene líneas *clear* asíncronas individuales $\overline{1CLR}$ y $\overline{2CLR}$. La burbuja de inversión frente al triángulo del símbolo lógico genérico de la figura 6.32b indica un dispositivo disparado por transición negativa. De manera análoga, el pequeño triángulo en cada entrada de reloj de la figura 6.32c es la notación estándar de IEEE para un *flip-flop* disparado por transición negativa.

Flip-flop JK 74276 cuádruple disparado por transición negativa

Suponga que su diseño requiere cuatro *flip-flops* JK. Considere el uso del SN74276 que aparece en la figura 6.32d. Éste cuenta con señales de control *preset* y *clear* comunes. Cada *flip-flop* lleva su señal de salida real Q a un *pin* del paquete del dispositivo. Cada *flip-flop* tiene entradas de excitación individuales J y $\overline{K}$, así como su propio reloj disparado por transición negativa.

Flip-flop 74111, dual con bloqueo de datos

El SN74111 que aparece en la figura 6.32e es una implantación especial del *flip-flop* JK. Tiene una característica de *bloqueo de datos* que combina un *latch* maestro disparado por transición positiva seguido de un esclavo disparado por pulso negativo. Es decir, el *latch* maestro opera en la transición delantera del

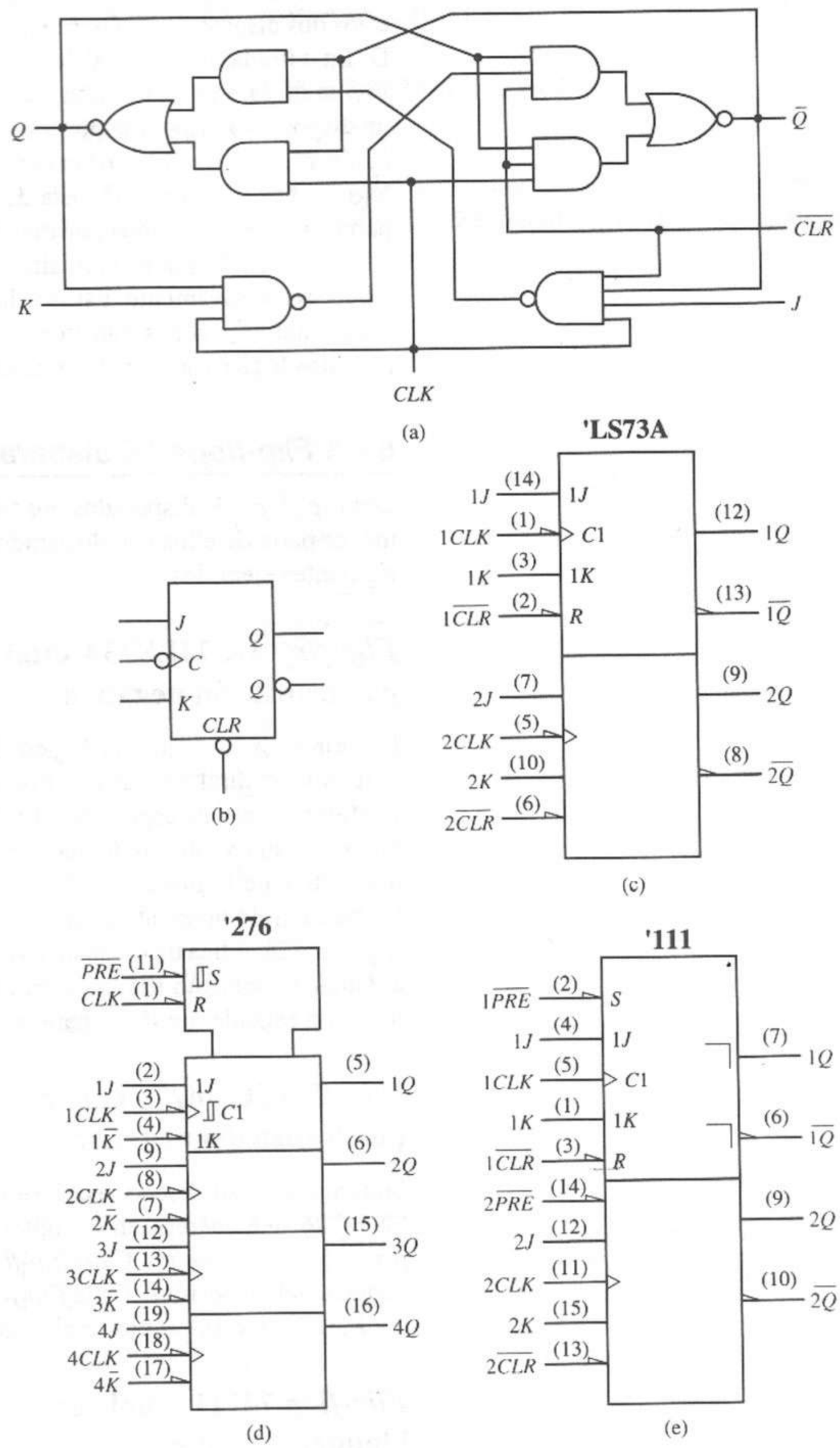


Figura 6.32 *Flip-flops* JK disparados por transición. **(a)** Diagrama lógico (SN74LS73A). **(b)** Símbolo lógico genérico (SN74LS73A). **(c)** Símbolo estándar de IEEE (SN74LS73A). **(d)** SN74276. **(e)** SN74111. *Fuente:* The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

reloj. Mientras el reloj permanece en alto, se ignoran los cambios posteriores en las entradas de excitación del maestro. Durante este tiempo, el *latch* esclavo está en el modo de retención, manteniendo el valor anterior del maestro. Cuando cae la señal del reloj, el nuevo valor en el maestro se envía a través de compuertas hasta el esclavo. Observe que esta combinación de controles de reloj se denota con la presencia de un pequeño triángulo frente a la entrada de reloj $C1$, lo que denota un *latch* maestro disparado por transición positiva, y un símbolo con una arista descendente adyacente a las salidas Q y $\overline{Q}$ del *flip-flop*, lo que denota que el esclavo se dispara mediante pulsos y cambia en la transición descendente del reloj. Este dispositivo tiene aplicaciones en diseños complejos en que las redes de distribución del reloj introducen retardos llamados *sesgos de reloj*. Podemos utilizar el SN74111 para minimizar el efecto del sesgo de reloj en el diseño de un sistema digital.

6.4.6 *Flip-flops* T

Flip-flop T disparado por transición

Un bloque de construcción de uso común en los circuitos lógicos secuenciales que cuenta los pulsos en una línea de señal es el *flip-flop* T (*trigger* o *toggle*, disparador o de alternancia). Aunque no está disponible como dispositivo TTL independiente, se utiliza con frecuencia para construir módulos contadores. El *flip-flop* T sólo tiene una señal de entrada de excitación, T, como lo muestra el símbolo lógico correspondiente de la figura 6.33a. La función de este dispositivo consiste en cambiar (alternar) su estado con cada transición en sentido negativo de su señal de entrada de excitación, como lo muestran la tabla de excitación y el diagrama de estados de las figuras 6.34a y b, respectivamente. Por tanto, la ecuación característica del *flip-flop* T disparado por transición es

$$Q^* = \overline{Q} \tag{6.16}$$

Una forma de visualizar la construcción de este dispositivo es considerar un *flip-flop* JK disparado por transición negativa, con sus entradas J y K altas. El dispositivo de la figura 6.33a se comporta como un *flip-flop* JK conectado como muestra la figura 6.33b. Ésta es la implantación de uso más frecuente, ya que hay varios *flip-flops* JK disponibles.

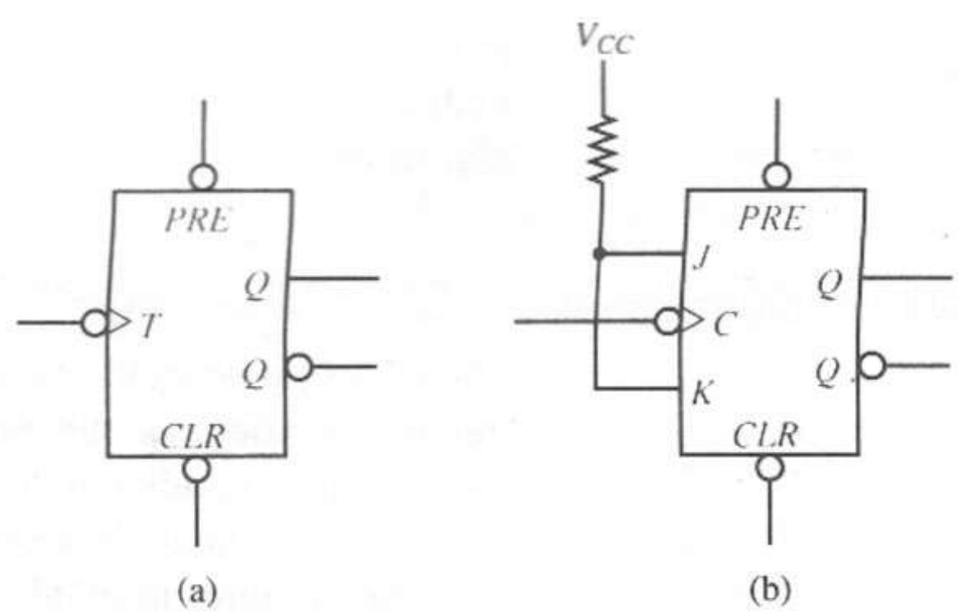
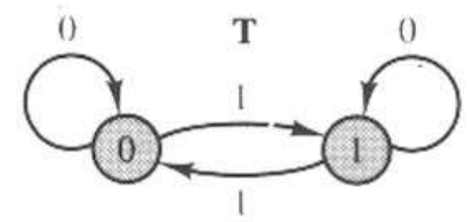


Figura 6.33 *Flip-flop* T disparado por transición negativa. **(a)** Símbolo lógico. **(b)** Equivalente funcional.

T	Q	Q^*	
↓	0	1	Alternancia
↓	1	0	Alternancia

(a)



(b)

Figura 6.34 Características del *flip-flop* T disparado por transición. **(a)** Tabla de excitación. **(b)** Diagrama de estados.

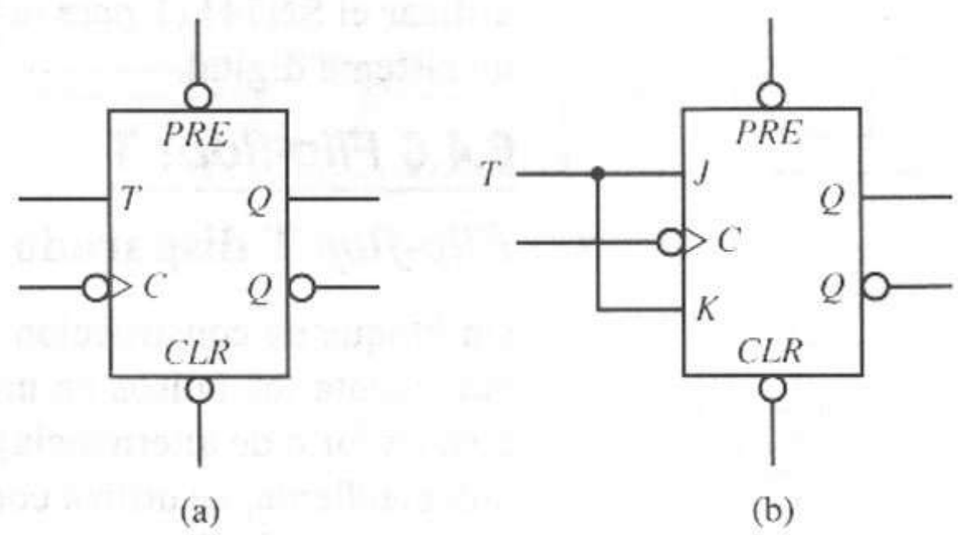


Figura 6.35 *Flip-flop* T con reloj. **(a)** Símbolo lógico. **(b)** Equivalente funcional.

T	Q	C	Q^*	
0	0	↓	0	Retención
0	1	↓	1	
1	0	↓	1	Alternancia
1	1	↓	0	

Figura 6.36 Tabla de excitación del *flip-flop* T con reloj.

Flip-flops T con reloj

Algunas versiones del *flip-flop* T operan bajo el control de los pulsos del reloj, como lo muestra la figura 6.35a. En este caso, el *flip-flop* alterna si $T = 1$ cuando el reloj hace una transición de alto a bajo y conserva su estado actual si $T = 0$ cuando el *flip-flop* está controlado por el reloj. El funcionamiento de un *flip-flop* T con reloj queda descrito por la tabla de excitación de la figura 6.36.

El circuito equivalente del *flip-flop* T con reloj, que aparece en la figura 6.35b, es sólo un *flip-flop* JK con entradas $J = K = T$, y su entrada C es controlada por la señal del reloj. La ecuación característica del *flip-flop* T con reloj se puede deducir de la ecuación del *flip-flop* JK, sustituyendo T por J y K de la manera siguiente:

$$\begin{aligned} Q^* &= J\bar{Q} + \bar{K}Q \\ &= T\bar{Q} + \bar{T}Q \end{aligned} \tag{6.17}$$

Para $T = 0$, la ecuación característica se reduce a $Q^* = Q$, que es la condición de retención, mientras que para $T = 1$, la ecuación característica es $Q^* = \overline{Q}$, que representa la condición de alternancia.

Otra variante del circuito *flip-flop* T con reloj aparece en la figura 6.37a. En este circuito, la señal de control T_c permite aplicar los pulsos del reloj de manera selectiva a la terminal de entrada T, de modo que cada pulso de reloj que llega a T provoca un cambio de estado del *flip-flop*. La figura 6.37b muestra un diagrama de tiempos detallado.

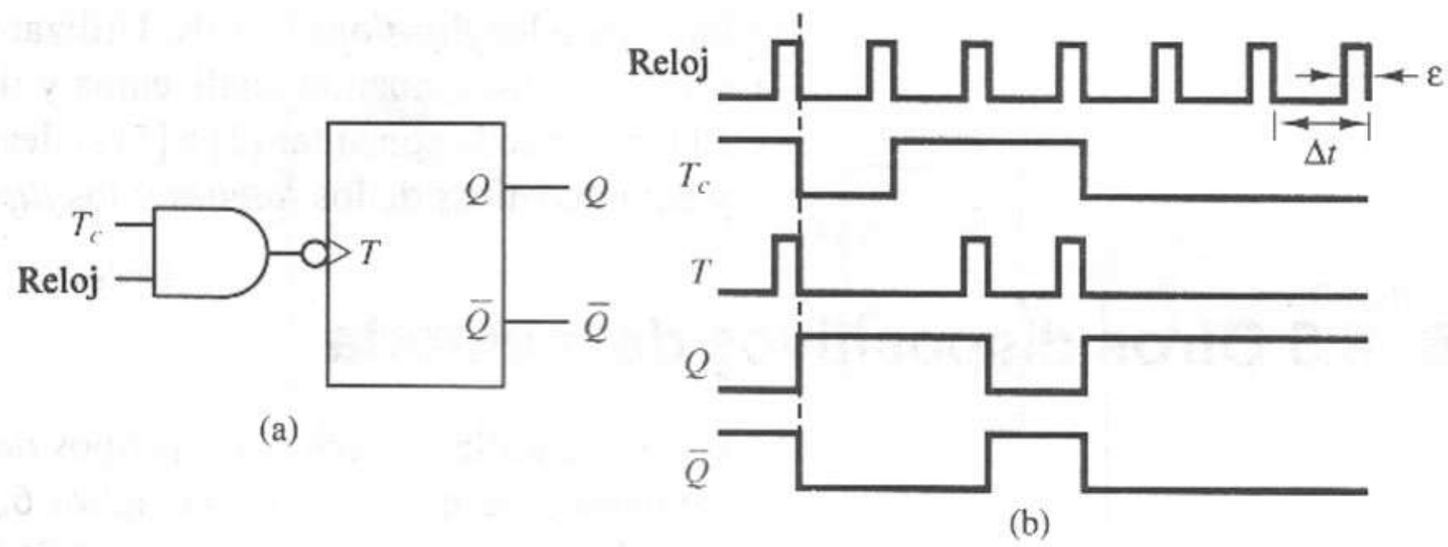


Figura 6.37 El *flip-flop* T con reloj. **(a)** Símbolo lógico. **(b)** Diagrama de tiempos.

6.4.7 Resumen de *latches* y *flip-flops*

En las secciones anteriores examinamos los dispositivos de memoria *latches* y *flip-flops*. Los circuitos *latch* se utilizan principalmente en situaciones en las que los datos se capturan desde líneas de señal y se guardan. El *latch* SR simple captura pulsos aleatorios en sus entradas S y R, pues cada pulso establece o reestablece el estado del *latch*. Los *latches* SR y D con compuertas cambian de estado sólo durante los instantes en que está habilitado el *latch*. Por tanto, los *latches* con compuertas se utilizan para capturar datos que llegan y se estabilizan antes del final de un pulso de activación.

Los *flip-flops* se utilizan principalmente para diseños de circuitos secuenciales en donde todos los cambios de estado deben sincronizarse con las transiciones de una señal de reloj. La mayor parte de estos circuitos utilizan *flip-flops* JK o D, según cuál de ellos necesite el menor número de compuertas para obtener las entradas de excitación para el diseño dado. Los *flip-flops* SR se utilizan poco, ya que los *flip-flops* JK proporcionan los mismos modos de operación y añaden el modo de alternancia, lo que elimina el problema de evitar la condición $S = R = 1$. Los *flip-flops* T se utilizan principalmente en el diseño de contadores.

La tabla 6.3 resume las ecuaciones características de los diversos dispositivos *latch* y *flip-flop* analizados en este capítulo. Puesto que los *flip-flops* disparados por pulsos tienen la misma ecuación característica que los *flip-*

TABLA 6.3 RESUMEN DE LAS CARACTERÍSTICAS DE LOS *LATCHES* Y LOS *FLIP-FLOPS*.

Dispositivo	Ecuación característica
Latch SR	$Q^* = S + \bar{R}Q$
Latch SR con compuertas	$Q^* = SC + \bar{Q}R + \bar{C}Q$
Latch D	$Q^* = DC + \bar{C}Q$
Flip-flop SR	$Q^* = S + \bar{R}Q$
Flip-flop D	$Q^* = D$
Flip-flop JK	$Q^* = \bar{K}Q + J\bar{Q}$
Flip-flop T (disparado por transición)	$Q^* = \bar{Q}$
Flip-flop T (con reloj)	$Q^* = T\bar{Q} + \bar{T}Q$

flops correspondientes disparados por transición, sólo hay una entrada en la tabla para los *flip-flops* D y JK. Utilizaremos estas ecuaciones características en otros capítulos, cuando analicemos y diseñemos varios circuitos secuenciales. El lector puede consultar [2] a [5] si desea mayor información acerca del diseño y características de los *latches* y los *flip-flops*.

6.5 Otros dispositivos de memoria

Podemos utilizar muchos otros tipos de dispositivos de almacenamiento como elementos de memoria en la figura 6.1b, como los núcleos magnéticos, los capacitores, las películas magnéticas, los elementos de criotrón superconductores y los relevadores electromecánicos. Puesto que estos otros tipos de memorias se utilizan poco en las computadoras actuales, su explicación queda fuera del objetivo de este texto. El lector puede consultar [5] y [6] si desea mayor información acerca de las diversas tecnologías y dispositivos de memoria.

6.6 Circuitos de tiempo

Otra clase de dispositivos de uso frecuente muy relacionados con los *flip-flops* son los módulos *one-shot* y de temporización. A continuación resumiremos estos importantes circuitos.

6.6.1 *One-shots*

Los *one-shots* son multivibradores monoestables; es decir, circuitos de almacenamiento digital con un único estado estable. Son llevados en forma temporal a un estado transitorio mediante una transición $0 \rightarrow 1$ o $1 \rightarrow 0$ en sus entradas y permanecen en dicho estado durante un tiempo especificado por la constante de tiempo de una red RC en los *pines* del paquete del dispositivo. El funcionamiento genérico de estos dispositivos se describe en varios textos de electrónica digital; por ejemplo, [5]. Las características específicas de la familia de dispositivos TTL aparecen en el manual TTL [1]. Vea, por ejemplo, el SN74121, SN74122 y SN74123. Los SN74122 y SN74123 son redisparables. Esto significa que su ciclo de tiempo programado se reinicia cada vez que sus entradas tienen una transición. SN74121 no es redisparable; después de iniciar un ciclo de tiempo programado, ignora los cambios posteriores en sus entradas hasta que concluye su ciclo de tiempo programado.

6.6.2 El módulo temporizador 555

El módulo temporizador 555 es un dispositivo de memoria que se utiliza en muchas aplicaciones, pues se puede configurar para utilizarse como *one-shot* o como un multivibrador astable, u oscilante [7]. EL 555 contiene un *latch* SR, como muestra la figura 6.38. Las entradas S y R del *latch* son controladas por las salidas de dos comparadores analógicos $C1$ y $C2$. Una segunda entrada *reset*, $R1$, se puede controlar en forma directa. La salida de cualquier comparador es 1 lógico si su entrada superior tiene un voltaje mayor que su entrada inferior. Una entrada de cada comparador se mantiene con un voltaje fijo mediante un divisor de voltaje con tres resistencias. La entrada inferior del comparador $C1$ se mantiene en 2/3 V_{CC}, mientras que la entrada superior del comparador $C2$ está en 1/3 V_{CC}.

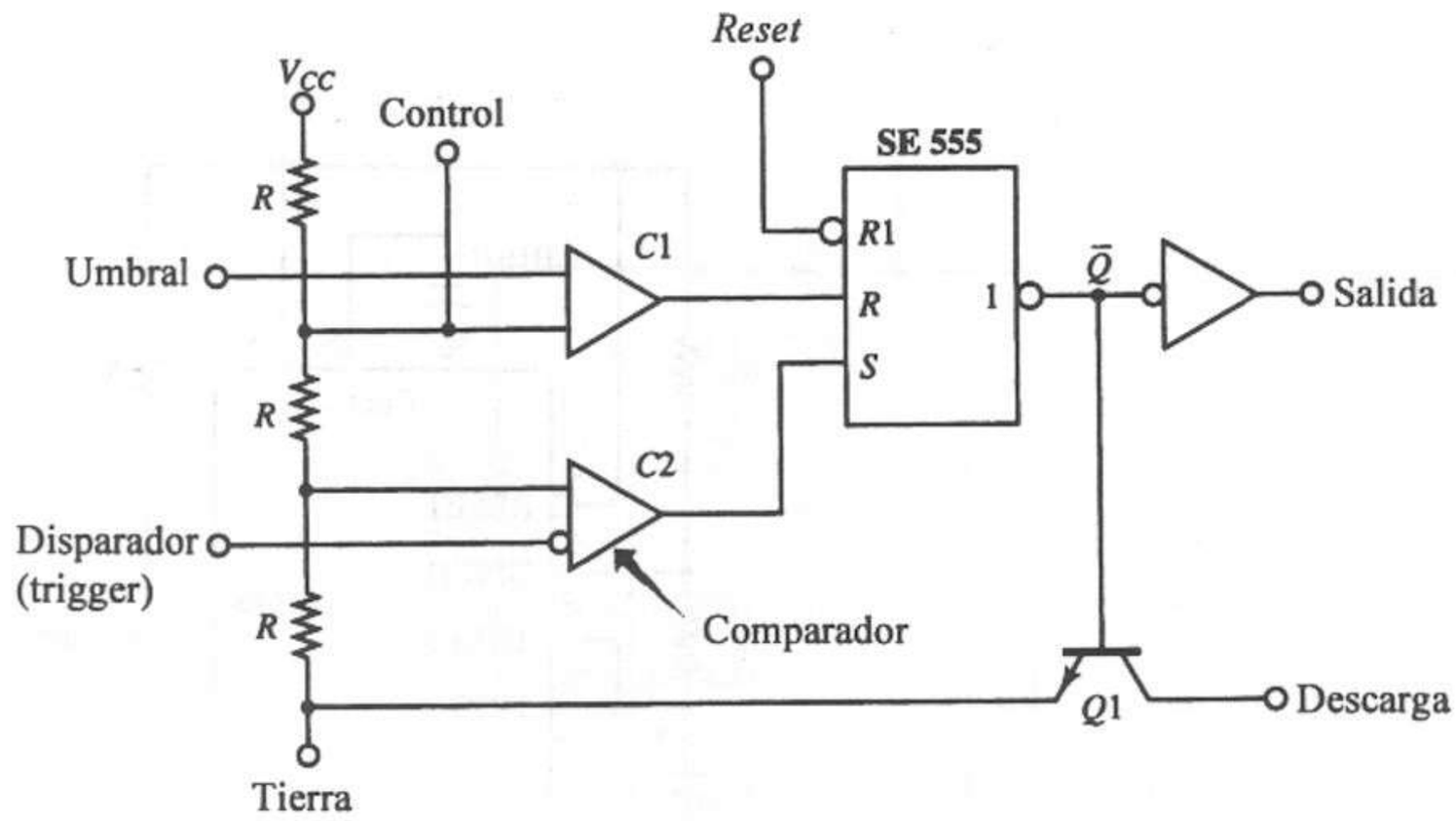


Figura 6.38 Módulo 555 (temporizador de precisión)

El *latch* se *establece* (*Salida* = 1) aplicando una señal 1 lógica al disparador. Esta señal se invierte a 0 lógico antes de entrar al comparador *C*2, haciendo que el voltaje en la entrada inferior sea menor que el valor 1/3 V_{CC} en la entrada superior, lo que hace que el comparador *C*2 aplique un 1 lógico a la entrada *S* del *latch*. El *latch* C1 se restablece llevando la entrada límite a un voltaje mayor que 2/3 V_{CC}, lo que hace que el comparador *C*1 aplique un 1 lógico a la entrada *R* del *latch*. Como en cualquier *latch* SR, hay que tener cuidado y no permitir que sea $S = R = 1$.

Funcionamiento astable

El funcionamiento astable se logra haciendo que el 555 se dispare a sí mismo; es decir, cambiando en forma alternada las entradas de disparo y de límite mediante un circuito *RC*, como se muestra en la figura 6.39. La frecuencia de oscilación es una función de las resistencias R_A y R_B y del capacitor C, y se calcula como sigue. El tiempo que tarda el capacitor en cargarse a través de R_A y R_B, y con ello el tiempo durante el cual la salida es 1 es igual a

$$t_H = 0.693(R_A + R_B)(C) \tag{6.18}$$

De manera análoga, el tiempo que tarda en descargarse el capacitor determina el tiempo durante el cual la salida es 0 y está dado por

$$t_L = 0.693(R_B)(C) \tag{6.19}$$

Al combinar las ecuaciones 6.18 y 6.19 obtenemos el periodo de oscilación T como

$$\begin{aligned} T &= t_H + t_L \\ &= 0.693(R_A + 2R_B)(C) \end{aligned} \tag{6.20}$$

La frecuencia de oscilación f es simplemente el inverso del periodo T definido en la ecuación 6.20, que es

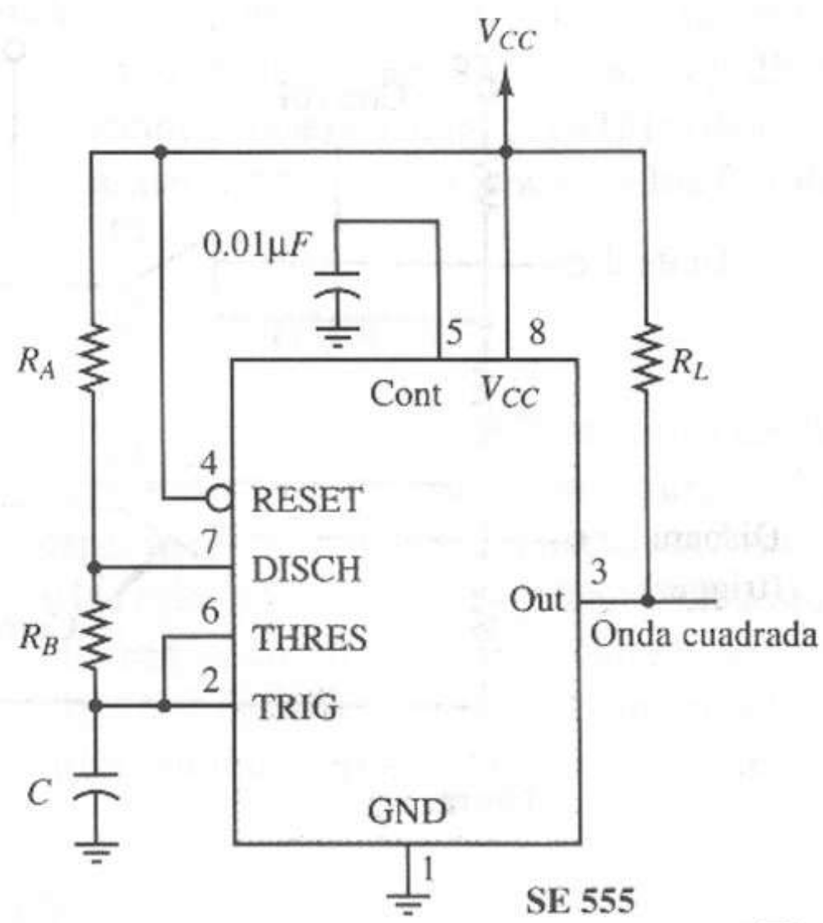


Figura 6.39 Módulo temporizador 555, configurado para el funcionamiento astable.

$$f = \frac{1.44}{(R_A + 2R_B)C} \tag{6.21}$$

Por ejemplo, para obtener $f = 45$ Hz, hay que elegir los siguientes valores para los componentes:

$$R_A = 100\ K\Omega$$
$$R_B = 100\ K\Omega$$
$$C = 3.125\ \mu F$$

Para los lectores que conocen de circuitos analógicos, el dispositivo funciona como sigue. Cuando la salida es 1, $\overline{Q} = 0$ y el transistor $Q1$ se *apaga*; es decir, se comporta como un circuito abierto. Esto hace que el capacitor C se cargue a través de las resistencias R_A y $\bar{R}_B$ hasta que el voltaje en los pines *Disparador/Límite* sea lo bastante alto como para que $S = 0$ y $R = 1$. Esto reestablece (*reset*) el *latch*, haciendo que la salida sea 0 y $\overline{Q} = 1$. En esta condición, el transistor $Q1$ se enciende; es decir, se comporta como un circuito cerrado, lo que provoca que el capacitor C se descargue a través de R_B y $\bar{Q}1$ hasta que el voltaje en los pines *Disparador/Límite* sea lo bastante bajo como para que los comparadores produzcan $R = 0$ y $S = 1$, estableciendo el *latch*. Esta operación continúa de manera indefinida.

Funciomaniento *one-shot*

Para el funcionamiento *one-shot*, queremos un elemento de almacenamiento que pueda dispararse a un estado dado, de modo que el estado regrese en forma automática a su valor definido por omisión después de cierto tiempo. Podemos hacer esto mediante un módulo temporizador 555 configurado como en la figura 6.40 para el funcionamiento *monoestable* (*one-shot*). El 555 se disparará mediante un pulso en su entrada *Disparador*, generando un pulso en su salida.

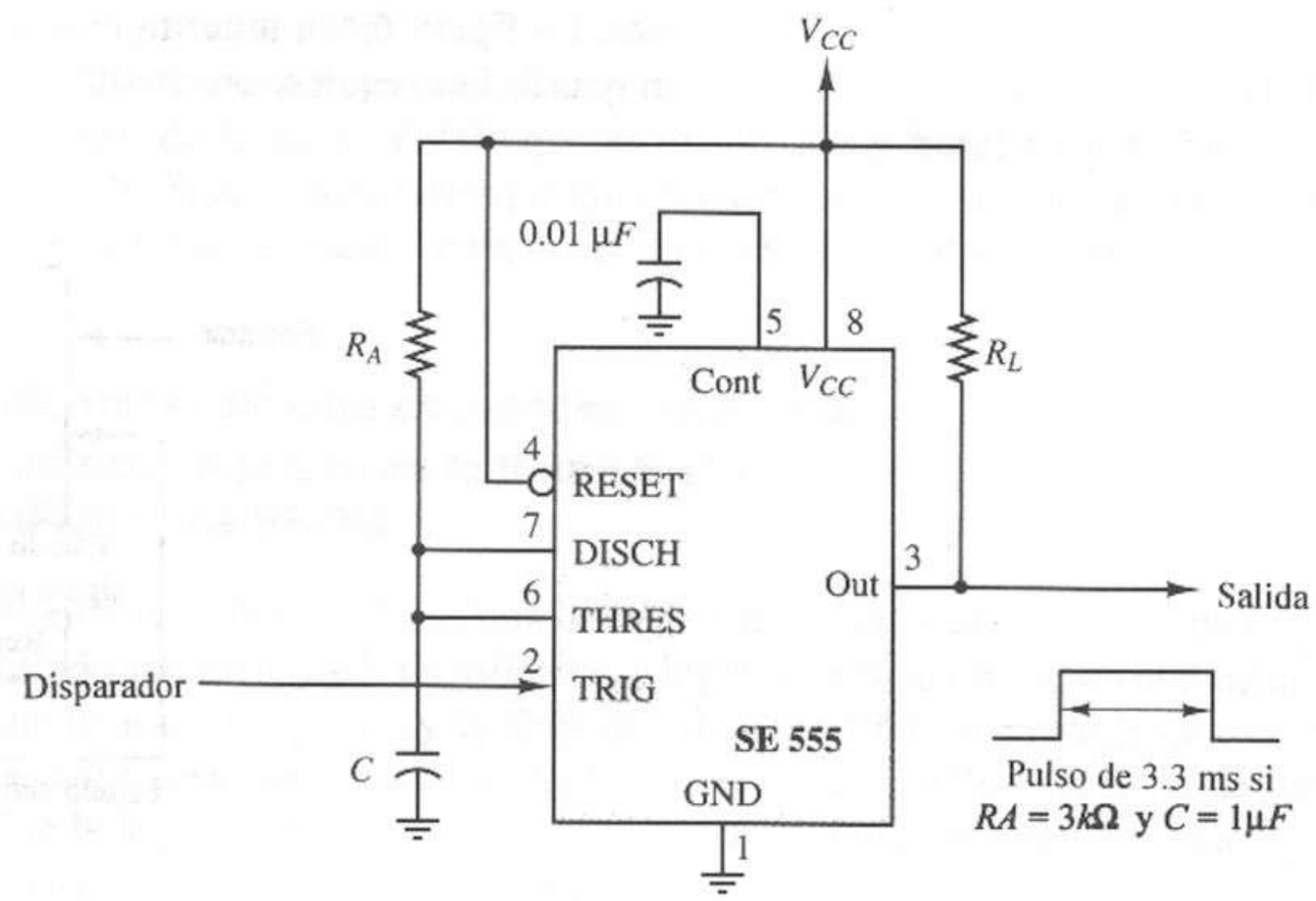


Figura 6.40 Diagrama lógico del *one-shot* 555.

El circuito 555 se configura para el funcionamiento monoestable mediante una resistencia R_A y un capacitor C, como se muestra en la figura 6.40. Con referencia al diagrama del 555 de la figura 6.38, *Salida* tiene el valor 1 si se aplica un pulso 0 lógico a la entrada de disparador. En la configuración de la figura 6.40, esto hace que el capacitor C se cargue a través de la resistencia R_A hasta que el voltaje en la entrada *Límite* sea lo bastante alto como para reestablecer (*reset*) el *latch*, regresando la salida a 0. Se supone que la entrada *Disparador* regresa a 1 lógico antes de que *Límite* reestablezca el *latch*.

Los valores de la resistencia R_A y el capacitor C determinan el tiempo que tarda el capacitor en cargarse y con ello la duración del pulso de salida. El ancho de pulso está dado por

$$t_W = 1.1(R_A)(C) \tag{6.22}$$

Así, por ejemplo, los valores $R_A = 3\ k\Omega$ y $C = 1\ \mu F$ producirían un pulso con una duración aproximada de 3.3 ms.

6.7 Creación rápida de prototipos de circuitos secuenciales

Ahora que hemos analizado los dispositivos de memoria, concluiremos este capítulo examinando en forma breve su uso para realizar circuitos secuenciales. La implantación de un circuito lógico secuencial requiere el diseño de los bloques lógicos combinatorios y de memoria del modelo de circuito secuencial de la figura 6.1b, dada una tabla de estados que describa al circuito. En los capítulos 8 y 9 examinaremos los métodos de diseño que minimizan el hardware de estos bloques. Sin embargo, podemos crear una implantación sencilla de cualquier

circuito secuencial mediante un registro de memoria que realice los bloques de memoria junto con dispositivos de memoria programable exclusiva para lectura (PROM), descritos en el capítulo 5, que realicen los bloques lógicos combinatorios. La figura 6.41a muestra una implantación PROM típica. Este método es muy útil si se requiere un circuito prototipo con rapidez.

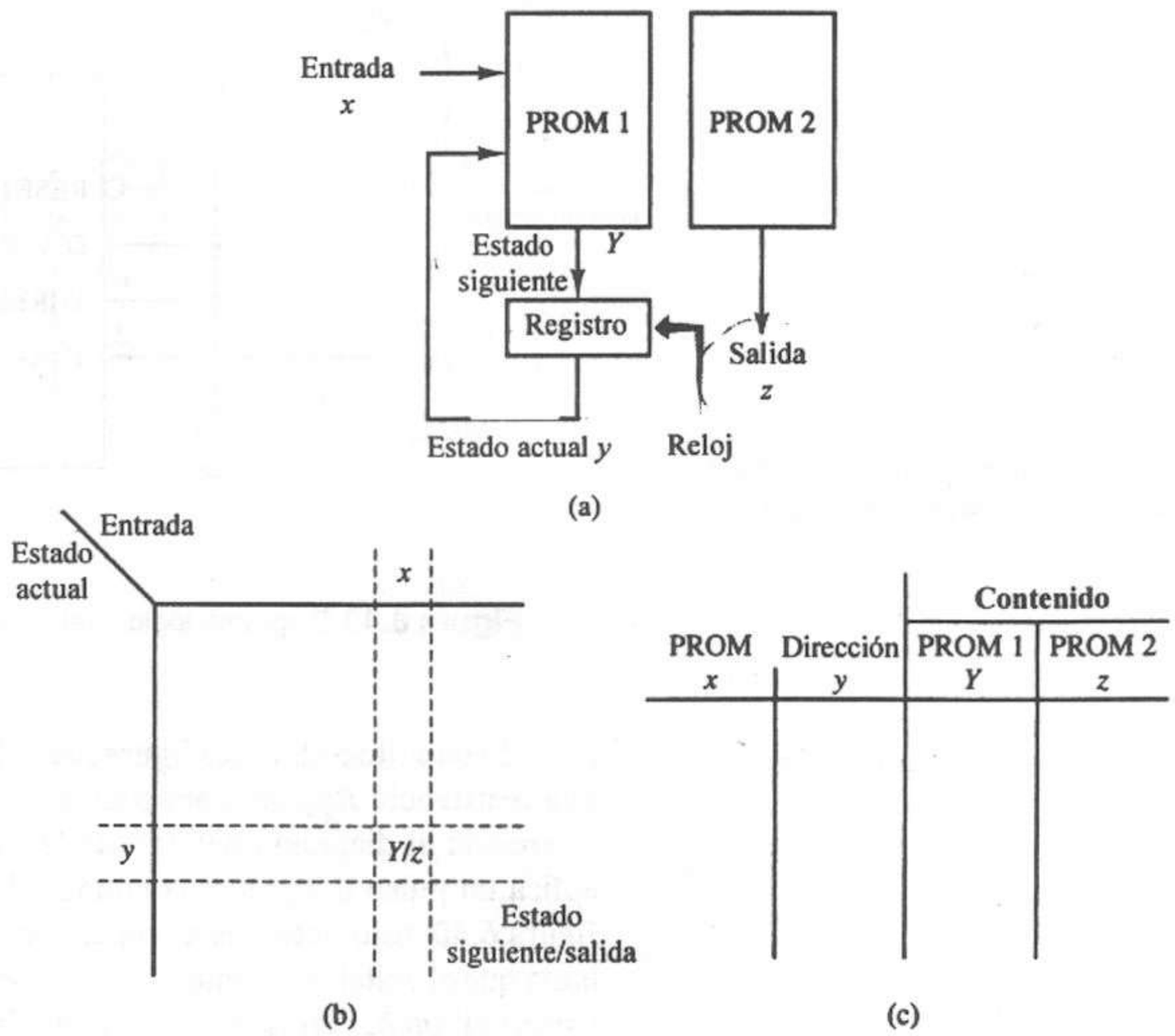


Figura 6.41 Circuitos secuenciales PROM. **(a)** Estructura de un circuito secuencial PROM. **(b)** Formato de tabla de estados. **(c)** Formato de un circuito secuencial PROM.

Dado un circuito secuencial con n entradas $x_1,\ldots,x_n$, m salidas $z_1,\ldots,z_m$ y r variables de estado $y_1,\ldots,y_r$, recordamos de las ecuaciones 6.2 y 6.3 que el bloque lógico combinatorio debe realizar las ecuaciones

$$z_i = g_i(x_1, \ldots, x_n, y_1, \ldots, y_r) \qquad i = 1, \ldots, m \tag{6.23}$$

$$Y_i = h_i(x_1, \ldots, x_n, y_1, \ldots, y_r) \qquad i = 1, \ldots, r \tag{6.24}$$

donde g_i y h_i son funciones de conmutación. Las r variables del siguiente estado, Y_i, representan los valores a guardar en los r elementos de memoria al cambiar de estado.

Con referencia a la figura 6.41a, el primer PROM implanta las funciones lógicas combinatorias del siguiente estado, h_i, mientras que el segundo PROM está dedicado a las funciones lógicas de salida g_i. Para muchas aplicaciones, las dos PROM se pueden implantar en un único circuito.

La utilidad de este tipo de diseño reside en la programabilidad de los dispositivos PROM. Si queremos cambiar el comportamiento del estado, entonces

sólo borramos y reprogramamos una o ambas PROM. Si colocamos las PROM en *sockets*, no hay que rehacer el alambrado de nuestro circuito para lograr estos cambios. ¡Los diseñadores aprecian esta característica cuando tienen que crear un prototipo del diseño!

¿Cómo especificamos el contenido de las PROMs? Consideremos la tabla de estados de la figura 6.41b, que es el formato general ya presentado en la figura 6.2b. Si sólo reordenamos la información como en la figura 6.41c, podemos utilizar la tabla de estados para generar la tabla de verdad de las PROM.

EJEMPLO 6.2

Implantar el circuito secuencial cuya tabla de estados aparece en la figura 6.42a mediante una PROM

La lógica combinatoria tiene una línea de entrada (x) y dos entradas de *flip-flop* de estado actual (y_2 y y_1). La salida de la lógica combinatoria serán dos señales de siguiente estado (Y_2 y Y_1) y la línea de salida (z). Podemos reordenar esta tabla de estados para dar la tabla de verdad de la figura 6.42b. Esta tabla de verdad define la lógica combinatoria necesaria para implantar el circuito secuencial de este ejemplo.

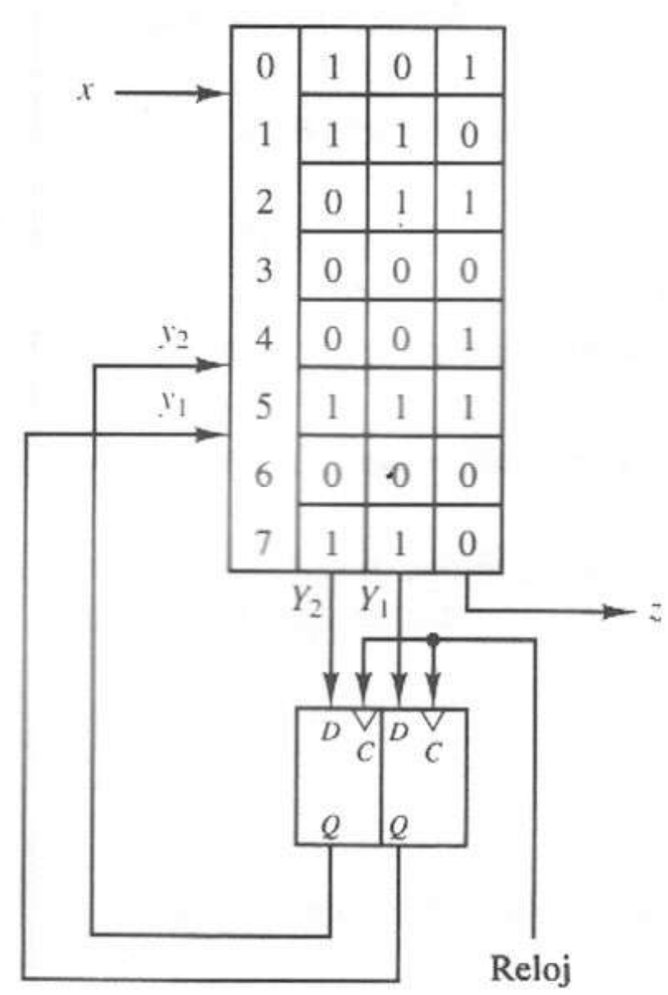
y_2y_1 \ x	0	1
00	10/1	00/1
01	11/0	11/1
10	01/1	00/0
11	00/0	11/0

Y_2Y_1/z

(a)

x	y_2	y_1	Y_2	Y_1	z
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

(b)



(c)

Figura 6.42 Ejemplo de circuito secuencial PROM. **(a)** Tabla de estados. **(b)** Tabla de verdad del PROM. **(c)** Implantación PROM.

Podemos utilizar una única PROM con ocho palabras de 3 bits cada una para implantar el circuito, como se muestra en la figura 6.42c. Dos *flip-flops* D completan la implantación.

EJEMPLO 6.3

Construir un circuito secuenciador que enumere los números primos menores que 256.

La figura 6.43 muestra la implantación PROM de este circuito. Observe que el circuito sólo utiliza algunas filas de la estructura de la PROM. Si el secuenciador comienza en el estado 2, pasa al estado 3, luego al 5, luego al 7, luego al 11, luego al 13, luego al 17, etcétera, hasta llegar al estado 251. El siguiente número primo mayor que 251 es mayor que 256, por lo que el secuenciador regresa la unidad al estado 2 para repetir el ciclo. Pero, ¿cómo podemos iniciar en el estado 2? Una forma de lograrlo es programar el estado 2 en todas las posiciones no utilizadas de la PROM. Al encender el dispositivo, comenzará en un estado no conocido. Si es un estado ilegal, es decir, si no es un número primo, el primer pulso de reloj envía la unidad al estado 2 y se inicia la secuencia.

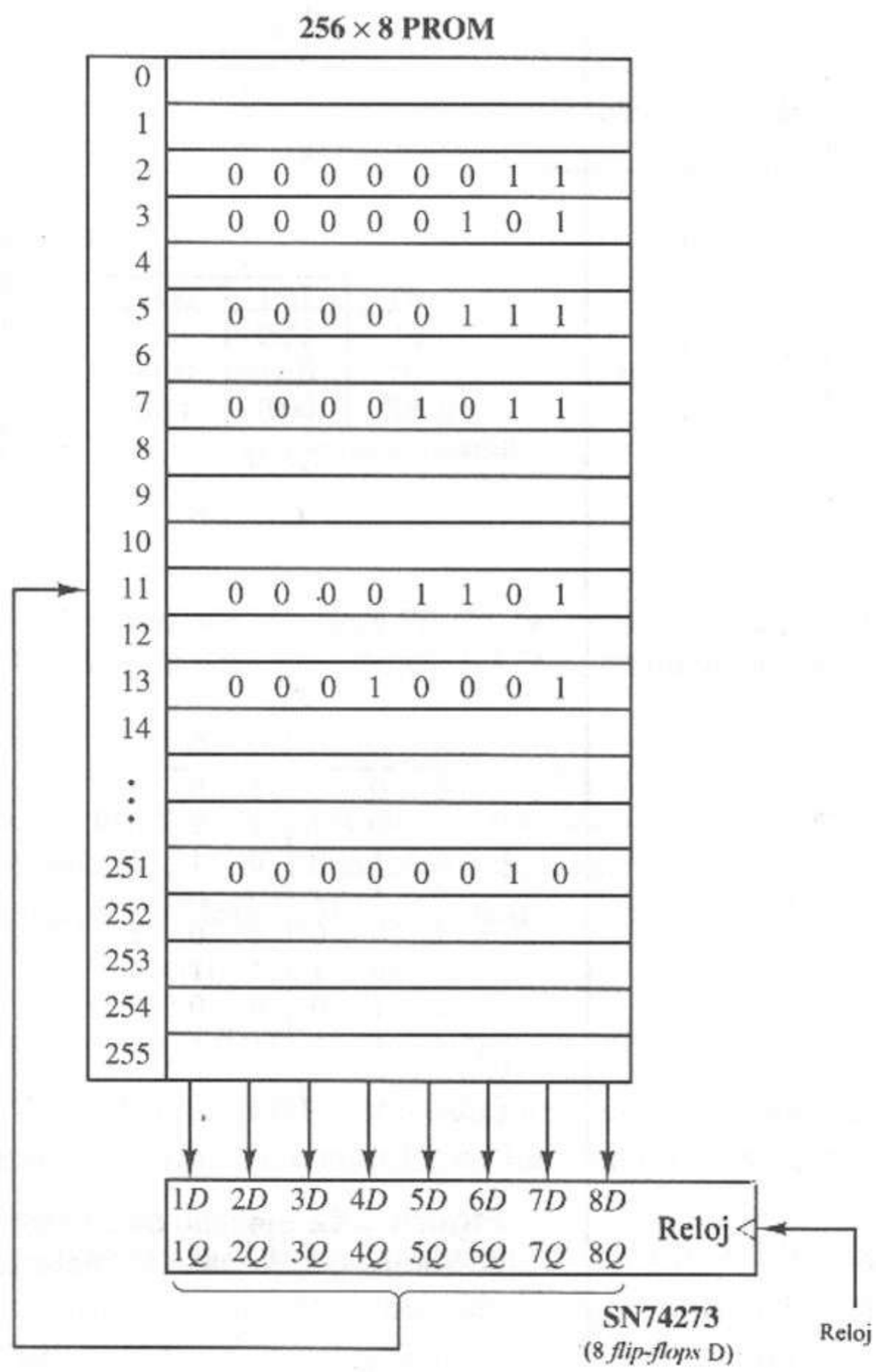


Figura 6.43 Implantación PROM de un secuenciador de números primos.

Observe que podemos implantar los ocho *flip-flops* del ciclo de retroalimentación de esta unidad con un circuito SN74273. Así, hemos implantado la unidad secuenciadora de primos con sólo dos circuitos.

El ejemplo anterior ilustra el uso de los dispositivos PROM para crear circuitos lógicos secuenciales. También se utilizan otros dispositivos lógicos programables en el diseño de los circuitos secuenciales; analizaremos esto en el capítulo 11.

6.8 Resumen

En este capítulo presentamos modelos para los circuitos secuenciales, incluyendo los diagramas lógicos, las tablas de estados y los diagramas de estados. Describimos con detalle varios elementos de memoria prácticos, como los *latches* y los *flip-flops*. En el capítulo 7 examinaremos los módulos secuenciales construidos mediante estos elementos. En los capítulos 8 y 9 examinaremos algunos métodos para el análisis y la síntesis de diversos circuitos secuenciales, síncronos y asíncronos construidos a partir de estos elementos de memoria.

REFERENCIAS BIBLIOGRÁFICAS

1. TEXAS INSTRUMENTS, *The TTL Data Book, Volume 2*. Dallas, Texas: Texas Instruments, Inc., 1988.
2. J. F. WAKERLY, *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice Hall, 1990, pp. 349–369.
3. R. H. KATZ, *Contemporary Logic Design*. Menlo Park, CA: Benjamin/Cummings, 1994, pp. 282–321.
4. J. P. HAYES, *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley, 1993, pp. 405–447.
5. G. E. WILLIAMS, *Digital Technology*. Chicago: Science Research Associates, 1977, pp. 166–169.
6. E. R. HNATEK, *A User's Handbook of Semiconductor Memories*. New York: Wiley, 1977.
7. S. A. R. GARROD AND R. J. BORNS, *Digital Logic: Analysis, Application & Design*. Philadelphia: Saunders College Publishing, 1991, pp. 412–423.

PROBLEMAS

6.1 Construya un diagrama de estados a partir de la siguiente tabla de estados. ¿Cuál es la ecuación lógica para la variable de salida z?

	x	
	0	1
A	$D/1$	$B/0$
B	$D/1$	$C/0$
C	$D/1$	$A/0$
D	$B/1$	$C/0$

6.2 Dada la siguiente tabla de estados, determine las secuencias de salida y de estados de la secuencia de entrada

$$x = 010101$$

si el circuito inicia en el estado A:

	x	
	0	1
A	$D/0$	$B/0$
B	$C/0$	$B/0$
C	$B/0$	$C/0$
D	$B/0$	$C/1$

6.3 Para el siguiente circuito secuencial, determine la secuencia de salida para la secuencia de entrada

$$x = 0010110101$$

si el estado inicial es A. Haga un diagrama de estados para este circuito.

	x	
	0	1
A	$B/0$	$C/1$
B	$C/1$	$B/0$
C	$A/0$	$A/1$

6.4 Deduzca el diagrama de estados y la ecuación característica del circuito *latch* de la figura P6.4.

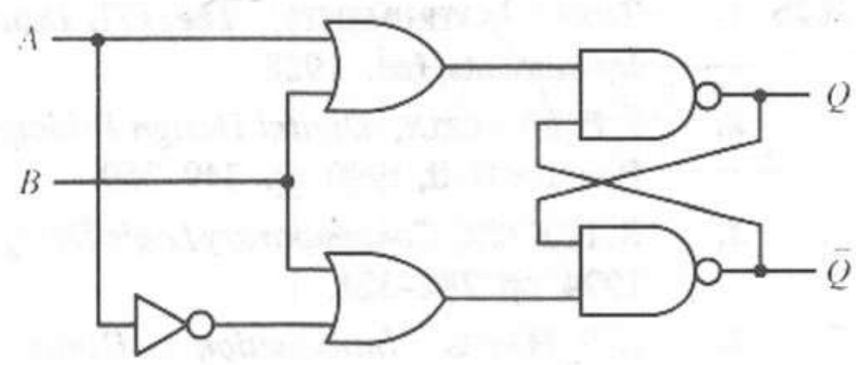


Figura P6.4

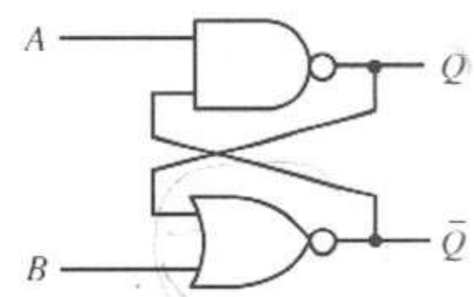
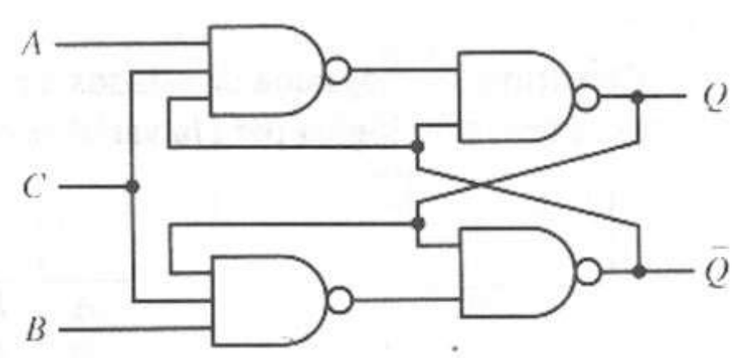


Figura P6.5

6.5 Determine la tabla de excitación del circuito *latch* de la figura P6.5 y describa su comportamiento con palabras.

6.6 ¿Es el circuito de la figura P6.6a un diseño válido *latch*? Explique. Si es un *latch*, complete la tabla de excitación de la figura P6.6b. ¿Se puede utilizar como un *latch* SR con compuertas? En tal caso, ¿de qué forma?



(a)

A	B	C	Q	Q^*	Modo
0	0	0	0	0	Sin cambio
0	0	0	1	1	Sin cambio

(b)

Figura P6.6

6.7 Construya los diagramas de estados de los siguientes:

(a) *Flip-flop* D
(b) *Flip-flop* SR
(c) *Flip-flop* T
(d) *Flip-flop* JK

6.8 Deduzca las ecuaciones características de estado que aparecen en la:

(a) ecuación 6.10 para los *latches* SR
(b) ecuación 6.12 para los *latches* D con reloj
(c) ecuación 6.15 para los *flip-flops* JK
(d) ecuación 6.17 para los *flip-flops* T con reloj

6.9 Dado el *flip-flop* JK de la figura P6.9a, complete el diagrama de tiempos de la figura P6.9b determinando la forma de onda de la salida Q.

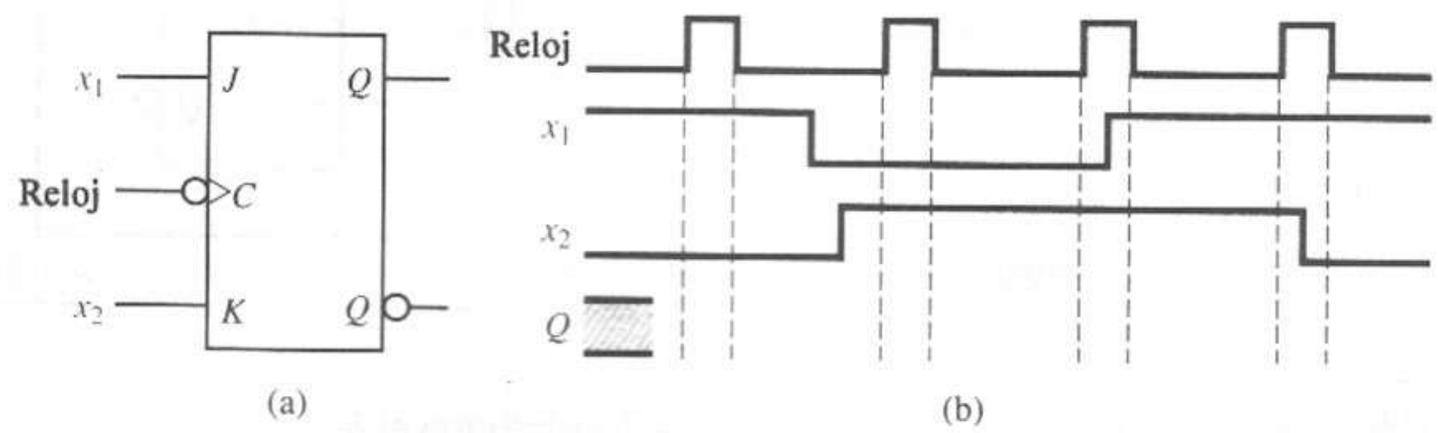


Figura P6.9 **(a)** *Flip-flop*. **(b)** Diagrama de tiempos.

6.10 Dado el *flip-flop* SR de la figura P6.10a, complete el diagrama de tiempos de la figura P6.10b determinando la forma de onda de la salida Q. Observe que el *flip-flop* se dispara en la transición positiva de la señal del reloj. Las entradas producen 2 veces la condición $S = R = 1$. ¿Conduce esto a un funcionamiento inestable? Explique.

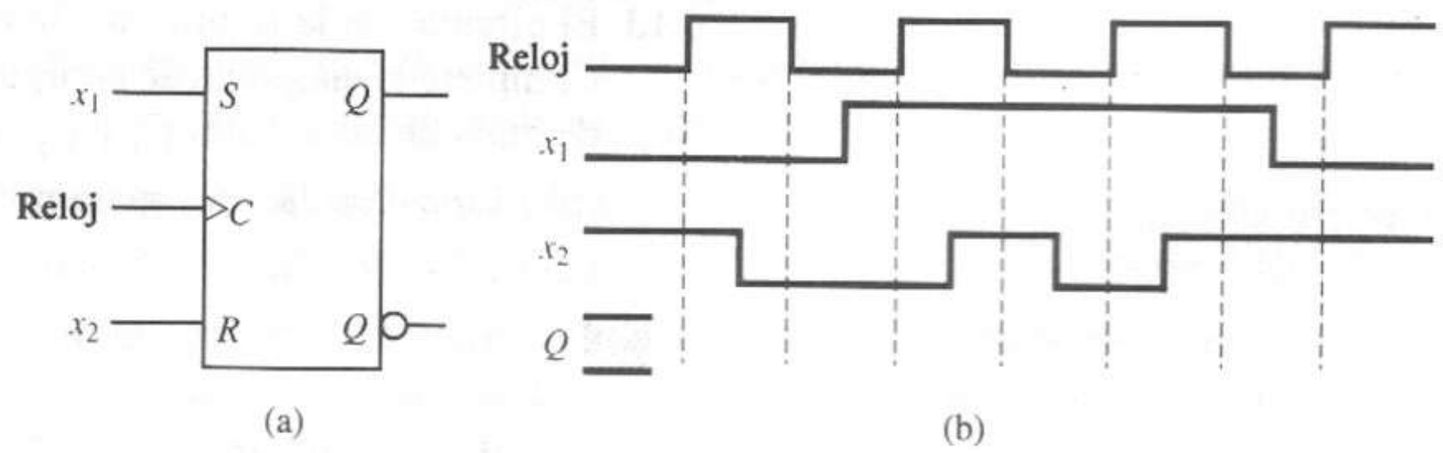


Figura P6.10 **(a)** *Flip-flop*. **(b)** Diagrama de tiempos.

6.11 Las formas de onda de la figura P6.11 se aplican a las entradas de un *flip-flop* JK SN7476. Complete el diagrama de tiempos trazando las formas de onda de las salidas Q y $\overline{Q}$ del *flip-flop*.

6.12 El circuito de la figura P6.12a contiene un *latch* D, un *flip-flop* D disparado por transición positiva y un *flip-flop* D disparado por transición negativa. Complete el diagrama de tiempos de la figura P6.12b trazando las formas de onda de las señales y_1, y_2 y y_3.

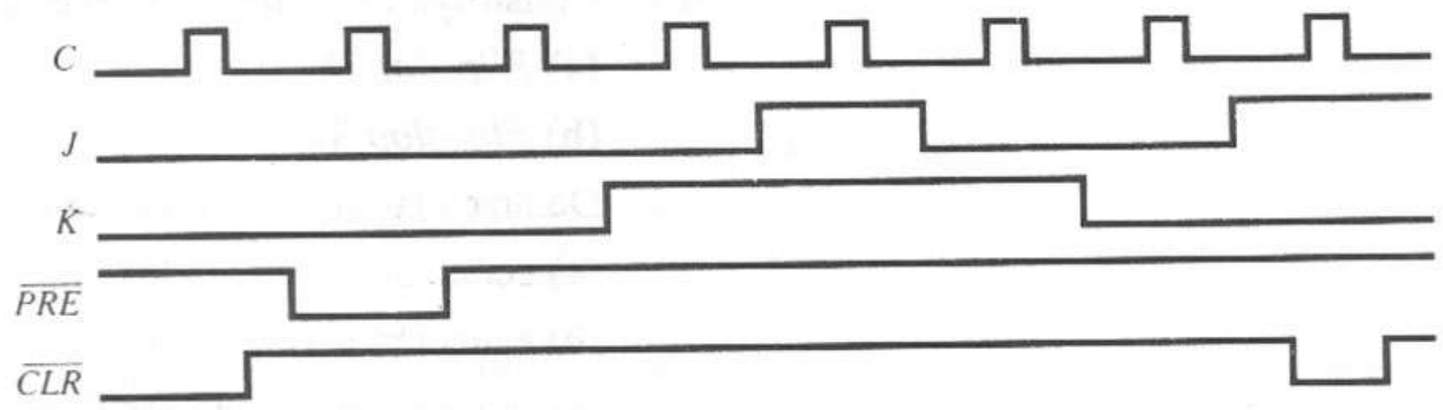


Figura P6.11

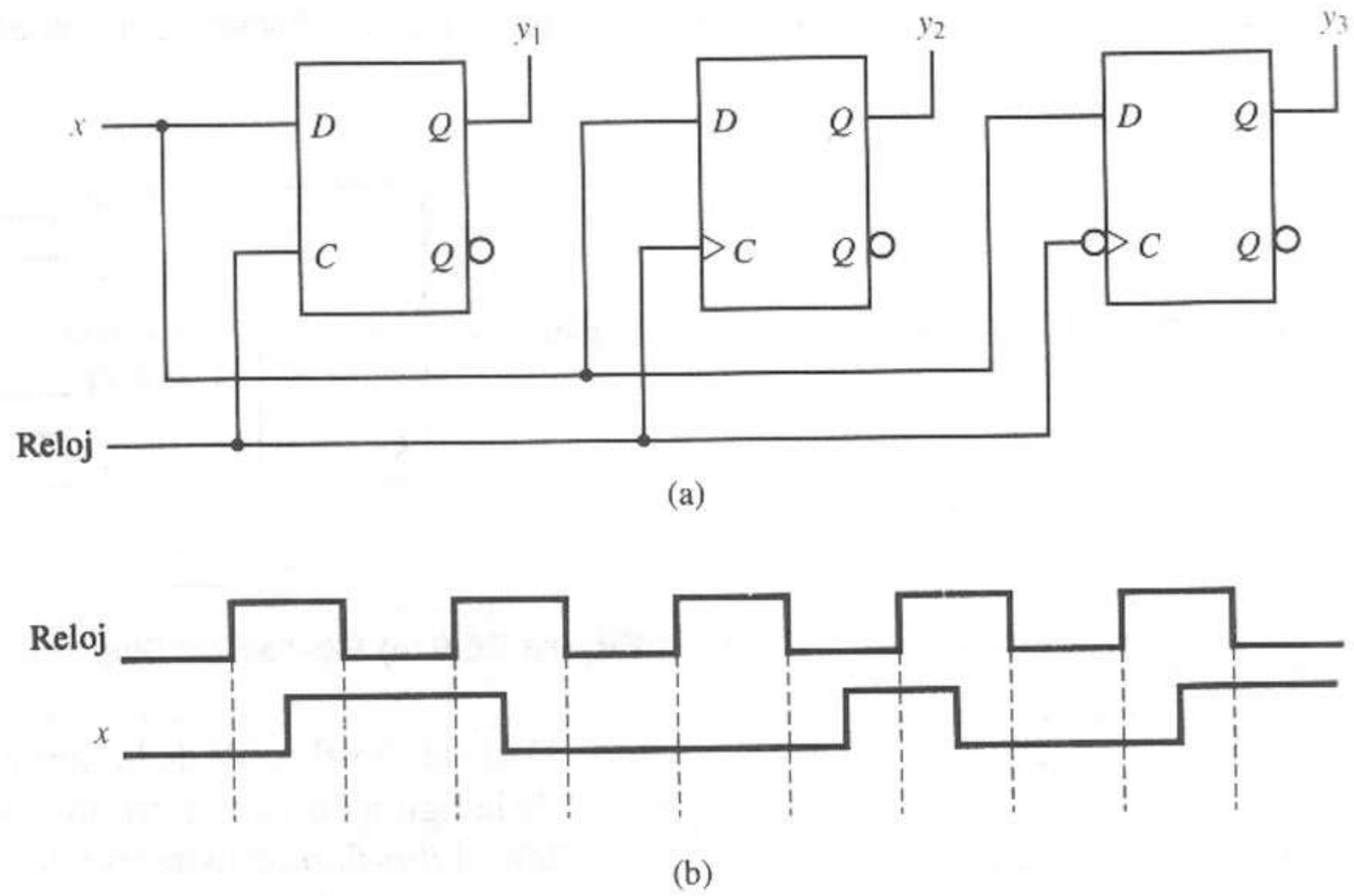


Figura P6.12 **(a)** *Flip-flop*. **(b)** Diagrama de tiempos.

6.13 El circuito de la figura P6.13a contiene un *flip-flop* JK y un *flip-flop* D. Complete el diagrama de tiempos de la figura P6.13b, trazando las formas de onda de las señales Q_1 y Q_2, suponiendo que:

(a) El *flip-flop* JK se dispara por transición negativa.

(b) El *flip-flop* JK tiene bloqueo de datos.

6.14 Complete la siguiente tabla de valores de excitación de un *flip-flop* necesarios para producir los cambios de estado indicados en el *flip-flop*, donde y indica el estado presente y Y el siguiente estado deseado del *flip-flop*.

Estado actual	Estado siguiente	*Flip-flop* JK		*Flip-flop* D	*Flip-flop* SR		*Flip-flop* T
y	Y	J	K	D	S	R	T
0	0						
0	1						
1	0						
1	1						

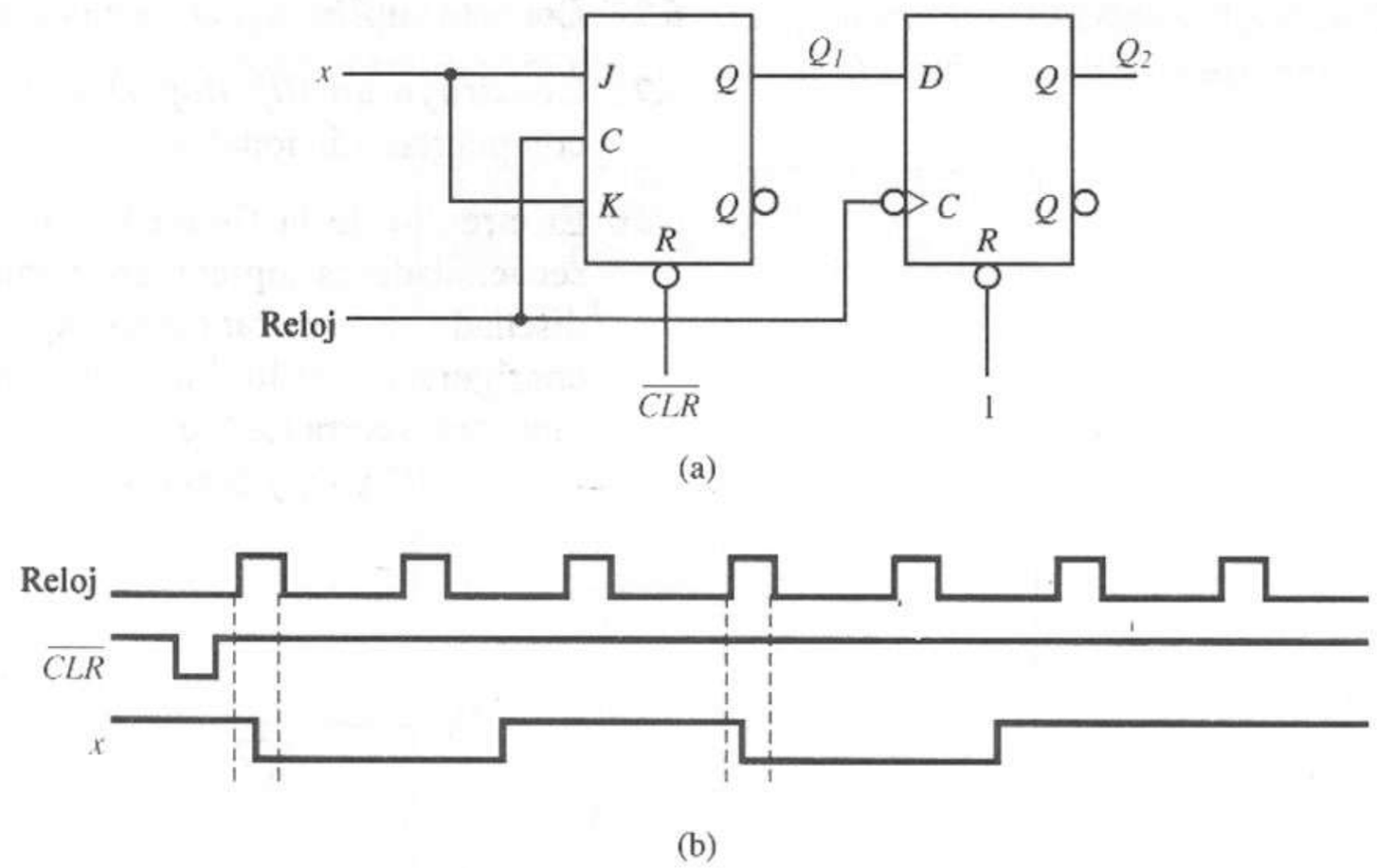


Figura P6.13 **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

6.15 Analice por qué la condición $S = R = 1$ conduce a una condición inestable para un *latch* SR.

6.16 Analice por qué la condición inestable $S = R = 1$ se evita en el *latch* de almacenamiento de los siguientes:

(a) *Latch* D

(b) *Flip-flop* JK

(c) *Flip-flop* T

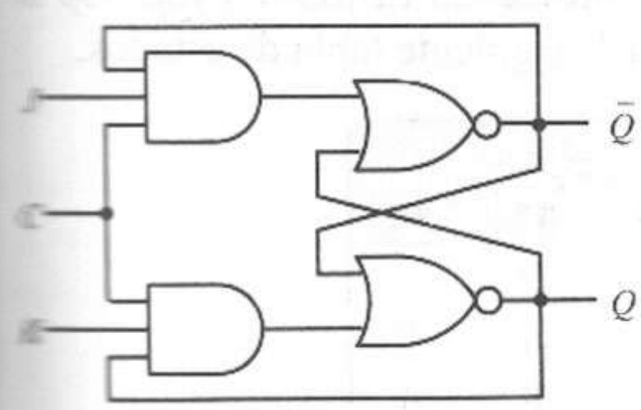


Figura P6.17

6.17 El circuito de la figura P6.17 pretende operar como un *latch* JK. Analice si este circuito es estable o no para la condición $J = K = C = 1$. Si el circuito es inestable, analice lo que podría hacer con la señal de reloj C para que el circuito opere de la manera deseada.

6.18 Diseñe un *flip-flop* JK maestro-esclavo con entradas *preset* y *clear* asíncronas que sólo utilice compuertas NOR.

6.19 Describa la diferencia operativa entre un *latch* D con reloj y un *flip-flop* D maestro-esclavo, desde el punto de vista de la salidas de los dispositivos.

6.20 Describa la forma en que parece operar un *flip-flop* maestro-esclavo como un dispositivo *disparado por transición*, desde el punto de vista de sus salidas externas.

6.21 Utilice el circuito de *latch* D libre de riesgos SN74116 de la figura 6.20c y diseñe un *flip-flop* D maestro-esclavo con entradas *preset* y *clear* asíncronas, $\overline{PRE}$ y $\overline{CLR}$, respectivamente.

6.22 Examine el circuito *flip-flop* D SN7474 de la figura 6.28a y describa la forma en que opera como un dispositivo disparado por transición.

6.23 Examine el circuito *flip-flop* JK SN7476 de la figura 6.27. Explique por qué las entradas $\overline{PRE}$ y $\overline{CLR}$ se conocen como entradas *asíncronas*, mientras que J y K son entradas *síncronas*.

6.24 Conecte un *flip-flop* D de modo que opere como un *flip-flop* T con reloj.

6.25 Construya un *flip-flop* D utilizando solamente un *flip-flop* JK, sin compuertas adicionales.

6.26 El circuito de la figura P6.26 es similar al utilizado en los circuitos secuenciadores lógicos programables disponibles en el comercio. Está diseñado para operar como un *flip-flop* JK o como un *flip-flop* D según la configuración de los interruptores SW_1 y SW_2. Determine la configuración (abiertos o cerrados) de los dos interruptores para el funcionamiento como *flip-flop* JK y D, y describa la forma de lograr este funcionamiento.

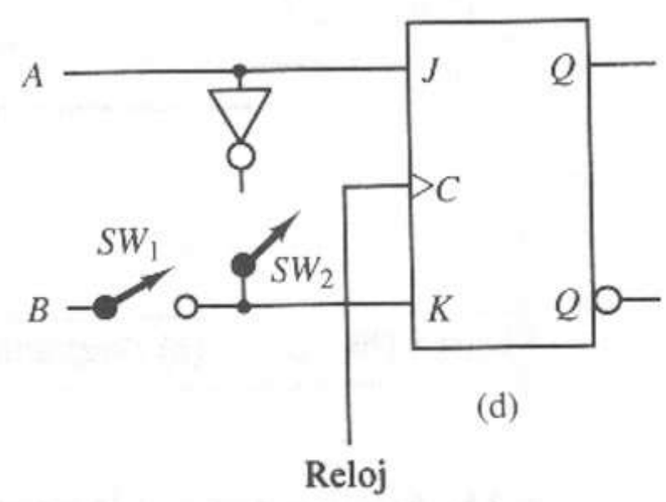


Figura P6.26

6.27 Diseñe un circuito secuencial síncrono, utilizando EPROM y *flip-flop* D disparado por transición, para implantar la siguiente tabla de estados:

	$x_1 x_2$ 00	01	11	10
$A = 00$	A	B	D	C
$B = 01$	B	C	A	D
$C = 10$	C	D	B	A
$D = 11$	D	A	C	B

Especifique el contenido de sus EPROMs en una tabla de números hexadecimales.

6.28 Diseñe un circuito secuencial síncrono, utilizando EPROMs y *flip-flops* D disparados por transición, para implantar la siguiente tabla de estados:

	$x_1 x_2 x_3$ 000	001	010	011	100	101	110	111
$A = 000$	A	B	H	C	B	B	B	D
$B = 001$	B	C	A	A	A	D	C	A
$C = 010$	C	D	B	E	D	B	D	B
$D = 011$	D	E	C	A	C	F	A	C
$E = 100$	E	F	D	G	F	B	F	H
$F = 101$	F	G	E	A	E	H	G	E
$G = 110$	G	H	F	A	H	B	H	F
$H = 111$	H	A	G	A	G	B	E	G

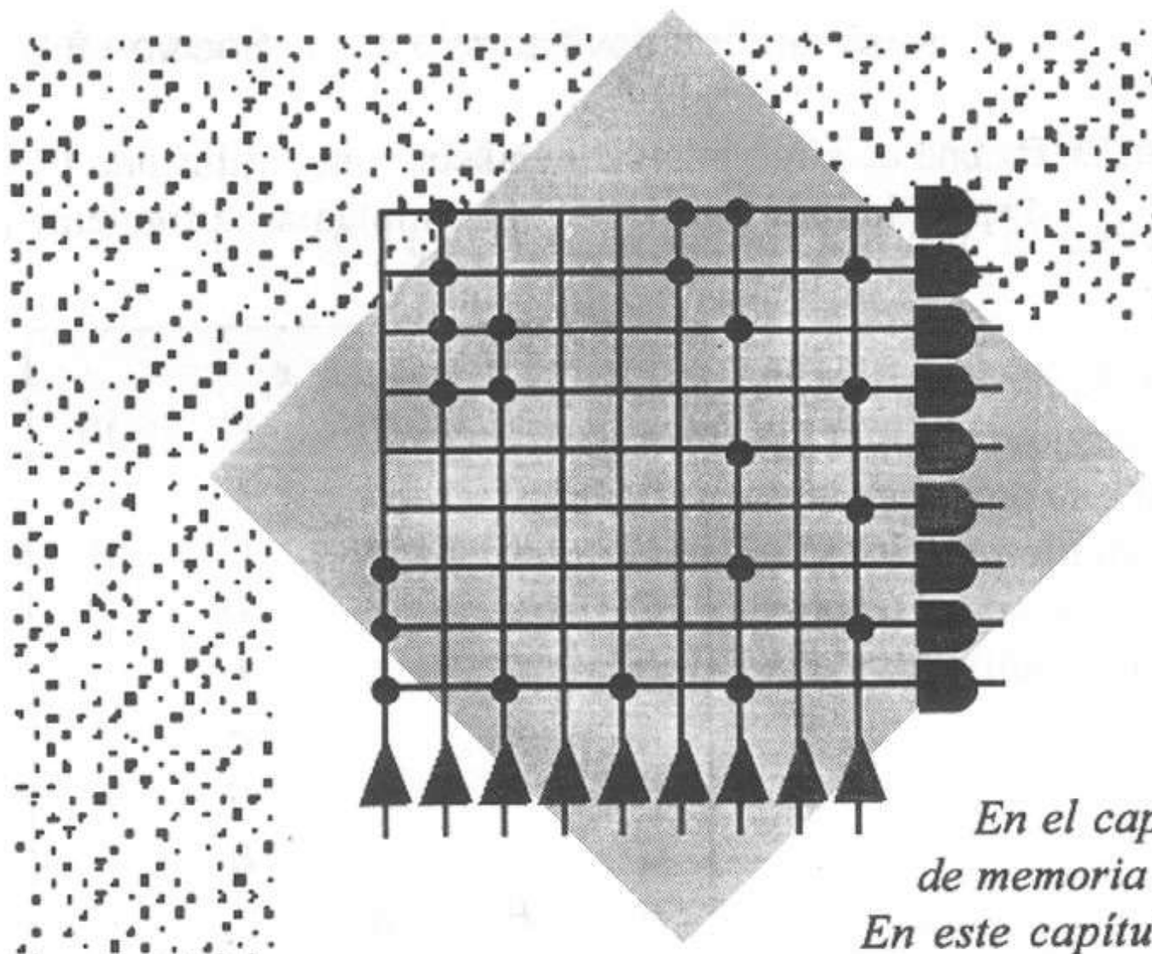
Especifique el contenido de su EPROM en una tabla de números hexadecimales.

6.29 Diseñe un circuito secuencial síncrono, utilizando EPROMs y *flip-flop* D disparados por transición, para implantar la siguiente tabla de estados:

	$x_1 x_2$ 00	01	11	10
A	*A*	*A*	*B*	*M*
B	*B*	*A*	*C*	*A*
C	*C*	*A*	*D*	*B*
D	*D*	*A*	*E*	*C*
E	*E*	*A*	*F*	*D*
F	*F*	*A*	*G*	*E*
G	*G*	*A*	*H*	*F*
H	*H*	*A*	*I*	*G*
I	*I*	*A*	*J*	*H*
J	*J*	*A*	*K*	*I*
K	*K*	*A*	*L*	*J*
L	*L*	*A*	*M*	*K*
M	*M*	*A*	*A*	*L*

Especifique el contenido de su EPROM en una tabla de números hexadecimales.

7



En el capítulo 6 examinamos diversos elementos de memoria binaria en forma de latches y flip-flops. En este capítulo investigaremos los módulos lógicos secuenciales, en donde se combinan los elementos lógicos combinatorios con los latches y flip-flops para realizar varias funciones de uso común. Estos módulos están disponibles como componentes TTL estándar y como funciones en las bibliotecas de diseño VLSI. Cada uno contiene flip-flops que se agrupan para formar un registro, que es el bloque básico de construcción de una computadora digital. Primero analizaremos los registros de corrimiento, que se utilizan para almacenar y manipular datos binarios. Después combinaremos estos registros de corrimiento con un sumador binario para producir un acumulador. Por último, examinaremos los contadores, que son configuraciones de flip-flops que generan series de números binarios.

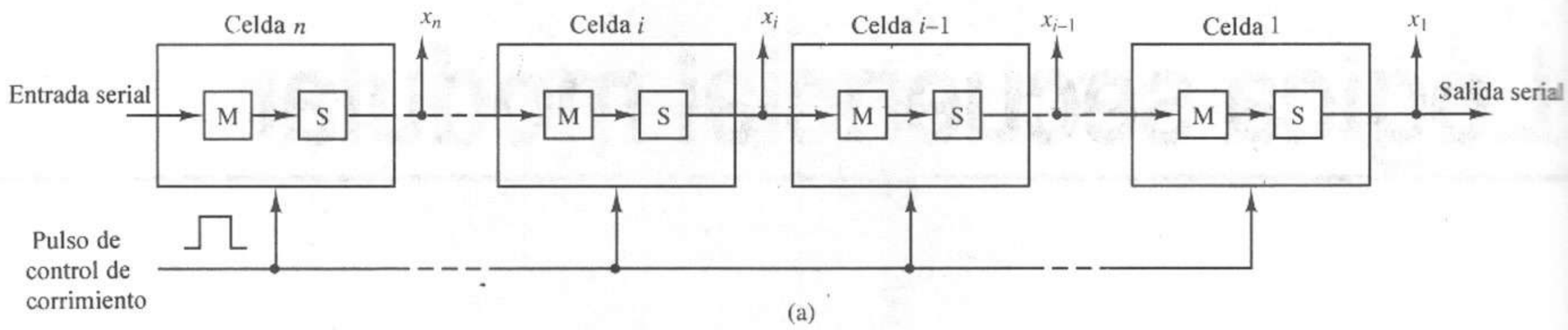
Todos los módulos de este capítulo se pueden diseñar con interconexiones directas de compuertas y elementos de memoria. En el siguiente capítulo examinaremos algunos métodos formales para el diseño de circuitos secuenciales.

Lógica secuencial modular

7.1 Registros de corrimiento

Un *registro de corrimiento* es un módulo lógico secuencial construido con *flip-flops* que controla las posiciones de los bits de datos binarios, recorriendo los bits a la izquierda o a la derecha. La figura 7.1 muestra un registro de corrimiento típico. El registro de corrimiento de n bits de la figura 7.1a utiliza n bits de datos binarios y se construye con un *flip-flop* maestro-esclavo. En el capítulo 6 presentamos el diseño y funcionamiento de los *flip-flops* maestro-esclavo. Cada *flip-flop* maestro-esclavo forma una *celda* del registro de corrimiento. Cada celda tiene un *latch* maestro y un *latch* esclavo, y contiene 1 bit de dato binario. El pulso de control del corrimiento para el registro es por lo general bajo y experimenta una rápida transición bajo-alto-bajo ($0 \rightarrow 1 \rightarrow 0$) para desplazar, o recorrer, el dato binario una posición a la derecha. Los bits de datos binarios residen normalmente en los *latches* esclavos. En la transición positiva ($0 \rightarrow 1$) del pulso de control de corrimiento, se transfieren los datos del *latch* esclavo de cada celda al *latch* maestro de la siguiente celda hacia la derecha. Observe que, en este momento, los datos en una celda son su valor antiguo (el valor de la salida binaria antes del pulso de corrimiento), que reside en el *latch* esclavo, y su nuevo valor (el valor de la salida binaria al terminar el pulso de corrimiento), que reside en su *latch* maestro. En la transición negativa ($1 \rightarrow 0$) del pulso de control de corrimiento, se transfiere el *latch* maestro de cada celda a su esclavo, dando un nuevo valor a su terminal de salida. Así, después de las dos transiciones del pulso de control de corrimiento, el bit binario de la celda X_i ha sido transferido a la celda X_{i-1}. En otras palabras, el número binario en el registro de corrimiento ha sido transferido una posición a la derecha. En consecuencia, decimos que éste es un registro de corrimiento *de entrada serial, de salida serial*. En la figura 7.1a hemos rotulado las terminales de entrada serial y de salida serial.

La figura 7.1b ilustra una implantación del registro de corrimiento de entrada serial, de salida serial, con *flip-flops* SR disparados por transición. Observe el símbolo lógico para el *flip-flop*. El reloj se rotula como CK. El triángulo denota que es una entrada disparada por transición, y la burbuja al frente indica que es sensible a una transición negativa ($1 \rightarrow 0$). Así, éste es un *flip-flop* SR *disparado por transición negativa*. El pulso de control de corrimiento (llamado *Shift*) es invertido por la compuerta NOT y aplicado a todas las entradas CK en forma simultánea. Estos *flip-flops* cambian al mismo tiempo cuando *Shift* pasa de bajo a alto ($0 \rightarrow 1$). En esta transición, los datos del *flip-flop* X_i se transfieren al *flip-flop* X_{i-1}. En consecuencia, esta configuración realiza la misma función de entrada serial, salida serial del registro en la figura 7.1a.



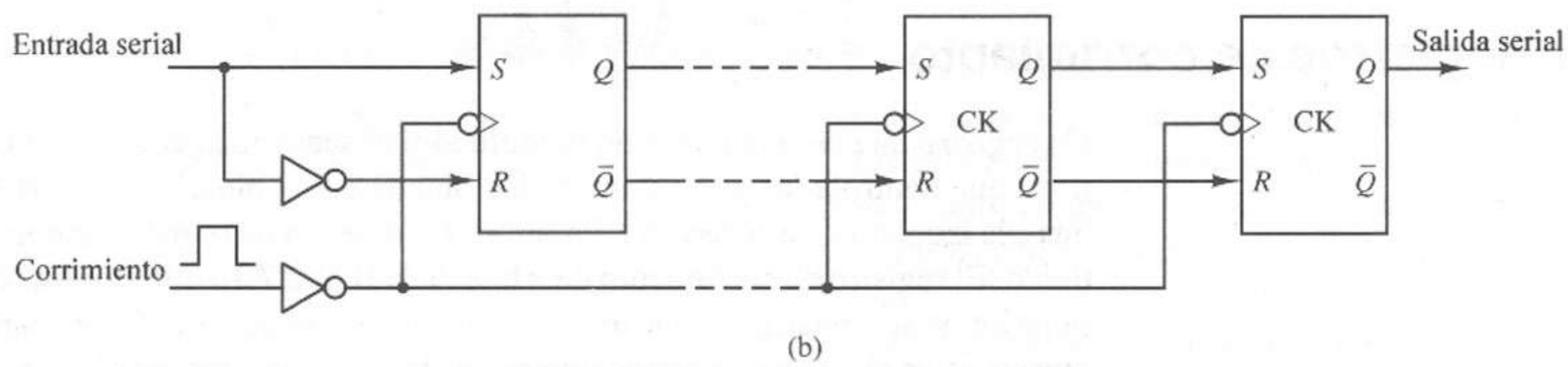


Figura 7.1 Registro de corrimiento de entrada serial, salida serial. **(a)** Registro de corrimiento con celdas maestro-esclavo. **(b)** Registro de corrimiento con *flip-flops* SR disparados por transición.

7.1.1 Un registro de corrimiento genérico

Consideremos ahora un elemento de registro de corrimiento *genérico*. Ver la figura 7.2. Las etiquetas del diagrama indican lo siguiente:

Entrada en paralelo **(Y_i, $i = 1,n$)**: una línea de entrada para cada *flip-flop* con datos por introducir en el registro

Salida en paralelo **(X_i, $i = 1,n$)**: una línea de salida que sale de la terminal *Q* de cada *flip-flop*

Pulso de corrimiento **(a veces rotulado *CLOCK*)**: un pulso en esta línea de control hace que los datos binarios del registro se desplacen una celda al mismo tiempo

Entrada serial: línea de datos que alimenta la primera celda del registro de corrimiento; entra un bit en cada pulso *Shift*

Salida serial: línea de datos de la terminal *Q* del último *flip-flop* del registro; sale un bit en cada pulso *Shift*

Control Clear (Limpiar): un pulso en esta línea hace que todos los *flip-flops* del registro asuman el valor 0 lógico

Control Preset (Preestablecer): un pulso en esta línea de control preestablece ciertos *flip-flops* con el valor 1 lógico, aquellos que tienen 1 lógico en su línea de entrada paralela Y_i.

El registro de corrimiento de la figura 7.2a puede operar de cuatro modos: de entrada serial y salida serial, entrada en paralelo y salida serial, entrada serial y salida en paralelo o bien entrada en paralelo y salida en paralelo. Los modos de entrada y salida serial requieren que los datos tengan un formato serial y

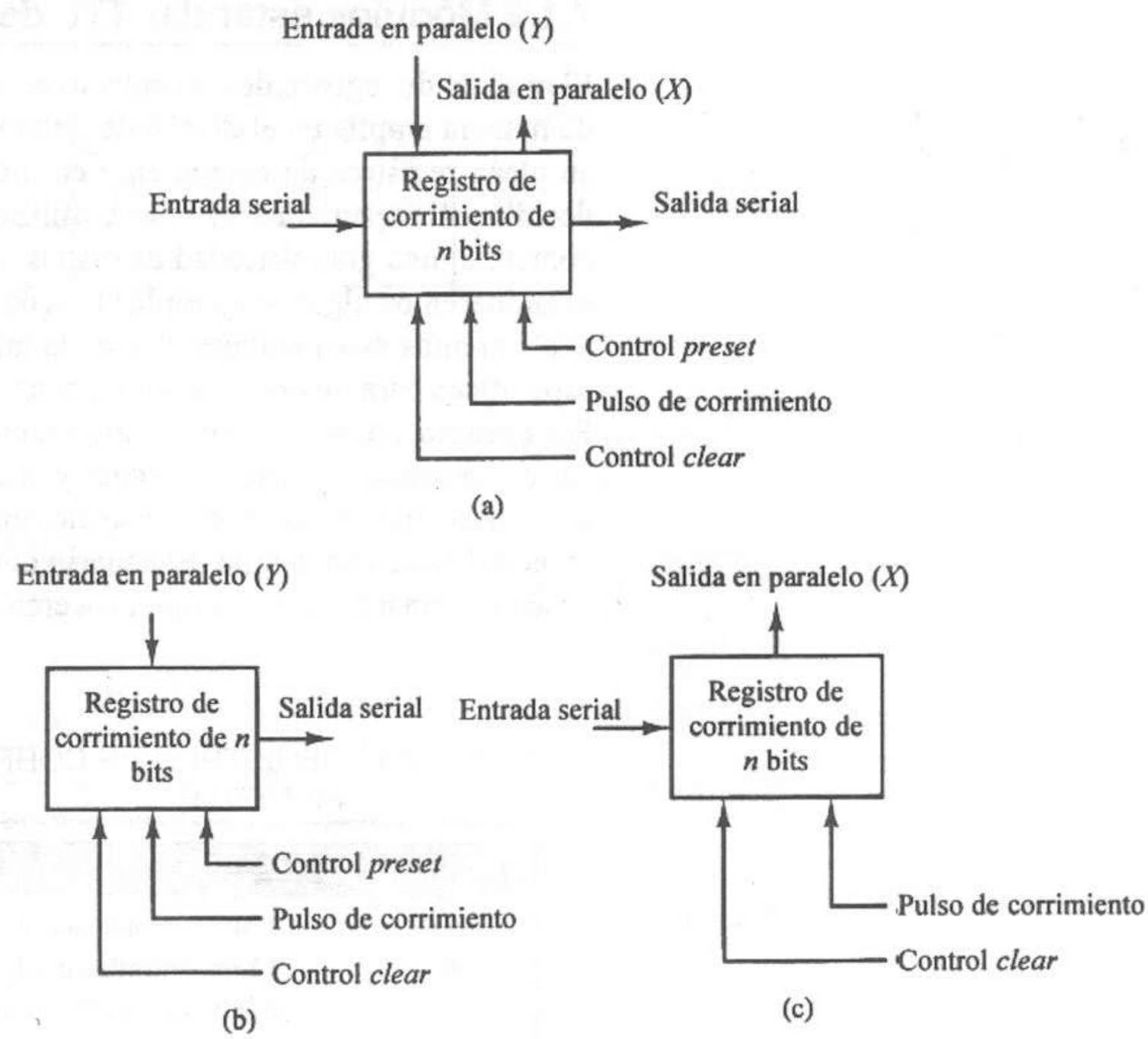


Figura 7.2 Registro de corrimiento genérico. **(a)** Símbolo lógico. **(b)** Entrada en paralelo, salida serial. **(c)** Entrada serial, salida en paralelo.

estén sincronizados con el *pulso Shift*. El registro de corrimiento sirve entonces como un retraso de tiempo de n pulsos para un flujo de datos seriales.

El modo entrada en paralelo y salida serial aparece en la figura 7.2b. El funcionamiento correcto de este modo requiere la siguiente secuencia de pulsos de control. Primero, hay que aplicar un pulso de *control Clear* para que todas las celdas del registro de corrimiento asuman el valor 0 lógico. Segundo, hay que conectar los datos paralelos con las líneas de entrada Y_i $i = 1,.., n$. Después aplicamos un pulso de *control Preset* para que algunas etapas del registro de corrimiento asuman el valor 1 lógico, como lo especifica el dato de entrada paralelo. Por último, se suministran n señales de *pulso Shift* a la unidad para generar un flujo serial de datos de salida.

Finalmente, consideremos el modo de operación entrada serial, salida en paralelo, que se muestra en la figura 7.2c. Para el funcionamiento correcto de este modo, hay que aplicar un pulso de *control Clear* a todas las celdas del registro de corrimiento para que tengan el valor 0 lógico. Después, aplicamos los datos *seriales de entrada* al registro, sincronizados con n señales de *pulso Shift*. Después de que el último *pulso Shift* asume el valor cero, los datos *paralelos de salida* están disponibles en las salidas X_i del *flip-flop*, $i = 1,n$.

7.1.2 Módulos estándar TTL de registro de corrimiento

El módulo de registro de corrimiento genérico descrito en la figura 7.2 se utiliza de manera amplia en el diseño de sistemas digitales. Los diseñadores digitales emplean registros de corrimiento en muchas y diversas aplicaciones. ¿Cómo decidir cuál registro de corrimiento utilizar? Se encuentran disponibles de manera comercial una gran variedad de dispositivos de registro de corrimiento. Ahora examinaremos algunos ejemplos típicos de la serie SN7400. Los módulos que analizaremos están enumerados en la tabla 7.1. Hemos elegido estos modelos específicos para ilustrar cuántas características diferentes han sido implantadas. Por ejemplo, ¿necesita usted un dispositivo bidireccional? ¿Cuál es la diferencia entre la retención de datos síncronos y la inhibición del reloj? ¿Necesita su diseño una carga síncrona o un *preset* asíncrono? En este capítulo examinamos estas características con detalle. El usuario debe consultar *The TTL Data Book* [1] si desea información más completa acerca de éstos y otros dispositivos.

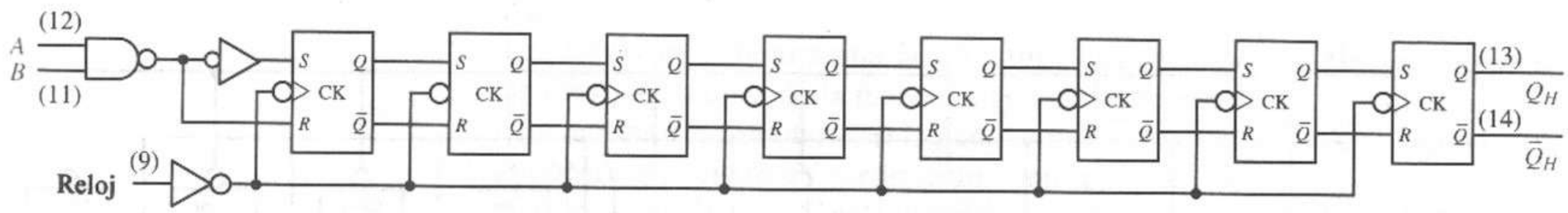
TABLA 7.1 REGISTROS DE CORRIMIENTO DE LA SERIE SN7400 [1]

Dispositivo	Características
7491A	8 bits, entrada serial, salida serial
7496	5 bits, entrada serial, salida serial, *preset* asíncrono, salida en paralelo, *clear* común
74164	8 bits, entrada serial, salida serial o salida en paralelo, *clear* común
74165	8 bits, entrada serial, salida serial, carga asíncrona, inhibición del reloj
74179	4 bits, entrada serial, salida serial, *clear* común, carga síncrona, salida en paralelo, retención síncrona de datos
74194	4 bits, bidireccional, entrada serial, salida serial, carga síncrona, salida en paralelo, inhibición del reloj, *clear* común

SN7491A

SN7491A es un registro de corrimiento de 8 bits, de entrada serial, salida serial. Es la versión de 8 bits de la figura 7.1b, con una característica adicional, una entrada serial con compuertas. Examine el diagrama lógico del dispositivo, que aparece en la figura 7.3a. Se realiza un AND con las entradas A y B en la terminal S del primer *flip-flop* SR. Esto permite utilizar una entrada como una fuente de datos serial y la otra como una línea de control de activación de datos. Observe que la línea de control de corrimiento de este dispositivo se rotula como *Clock*. Podemos limpiar el registro manteniendo en cero la línea de activación y pulsando la línea *Clock* ocho veces. Observe que los datos verdaderos y complementados están disponibles en la salida del octavo *flip-flop*.

La figura 7.3b muestra la tabla funcional para el dispositivo. Cuando ambas entradas A y B son altas, los bits de datos 1 lógico entran al dispositivo. Después de ocho pulsos de reloj (en el instante t_{n+8}), los datos altos llegan a las terminales de salida (Q_H y $\overline{Q}_H$). Si la entrada A o la entrada B es baja, los datos que llegan a la salida en el instante t_{n+8} tienen el valor 0 lógico.



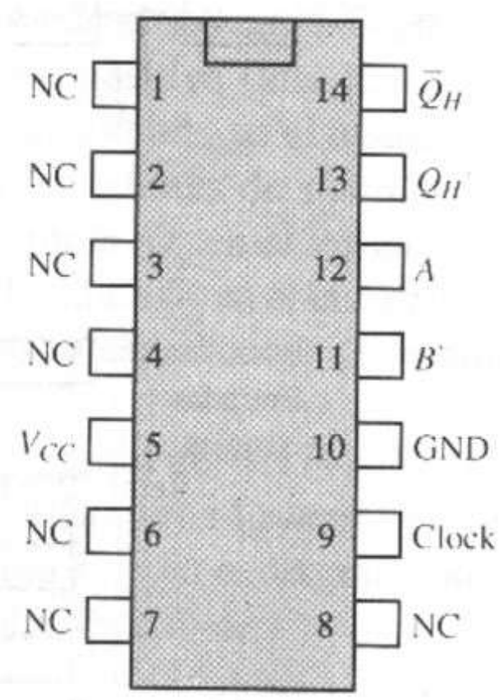
(a)

Entradas en tn		Salidas en t_{n+8}	
A	B	Q_H	$\overline{Q}_H$
H	H	H	L
L	×	L	H
×	L	L	H

t_n = tiempo de bit de referencia, reloj bajo

t_{n+8} = tiempo de bit después de 8 transiciones baja a alta del reloj

(b)



(c)

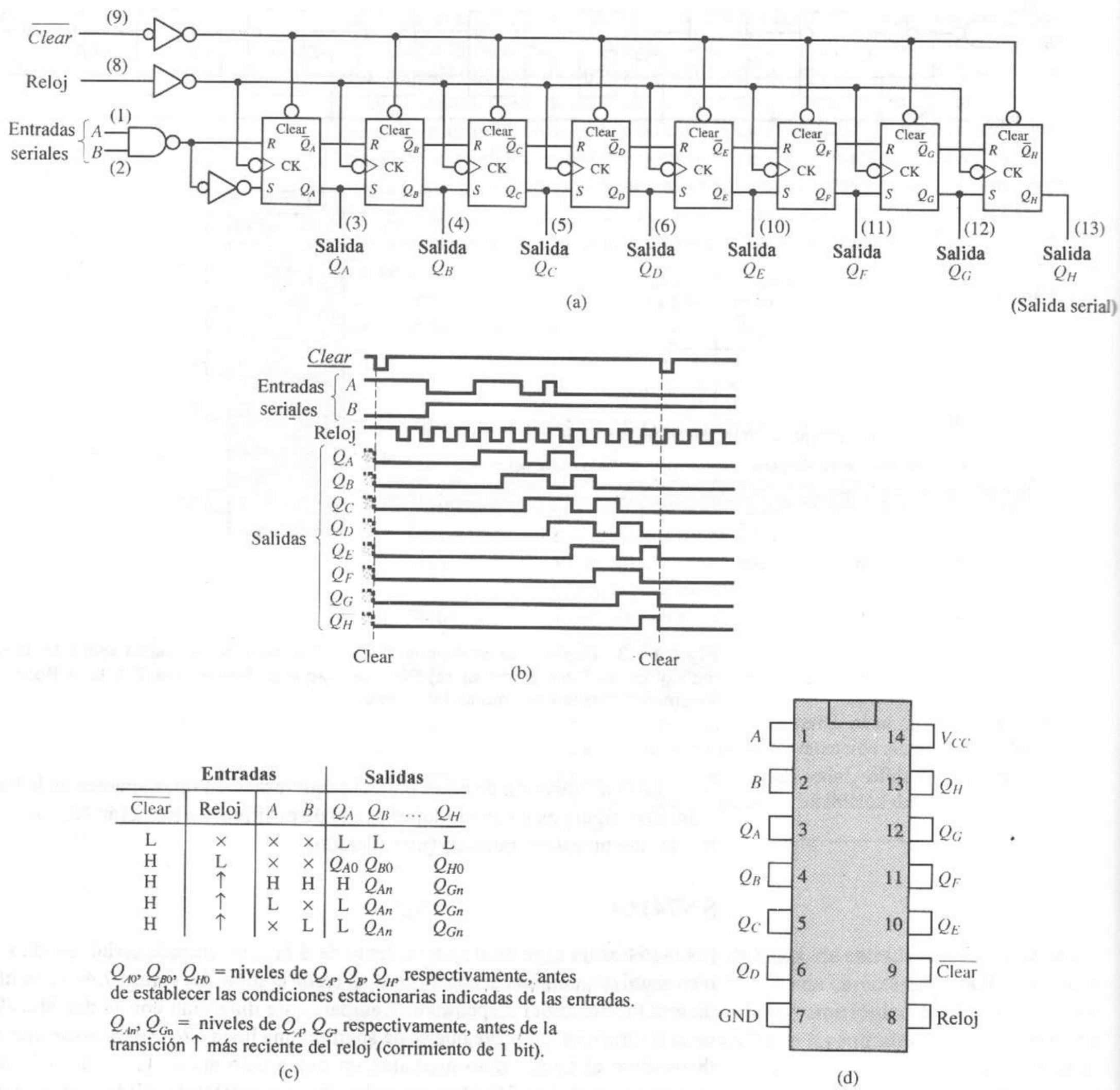
Figura 7.3 Registro de corrimiento SN7491A entrada serial, salida serial. **(a)** Diagrama lógico. **(b)** Tabla funcional. **(c)** *Pines* del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

La configuración de *pines* para el paquete dual en línea aparece en la figura 7.3c. Esta figura es una vista superior del dispositivo. La notación NC en un *pin* indica que no está conectado (no utilizado).

SN74164

SN74164 es un registro de corrimiento de 8 bits, de entrada serial, salida serial o en paralelo, con una señal de control *clear* común. La figura 7.4a presenta el diagrama lógico del dispositivo. Compare este diagrama con el del SN7491A de la figura 7.3a. Observe que se ha añadido una línea $\overline{Clear}$ para poder llevar el dispositivo al estado nulo mediante un pulso bajo activo ($1 \rightarrow 0 \rightarrow 1$). Otra característica del SN74164 es sus ocho líneas paralelas de salida, una para cada uno de los ocho *flip-flops* del dispositivo. En este dispositivo no se dispone del complemento de la última celda del registro, como en el caso del SN7491A.

Una forma de mostrar el comportamiento de este dispositivo es mediante un diagrama de tiempos. Ver la figura 7.4b. La secuencia comienza con un pulso en sentido negativo en la línea de entrada $\overline{Clear}$ para llevar al dispositivo al estado nulo. Después, se proporcionan al reloj una serie de pulsos. El dispositivo examina los valores en las líneas de entrada seriales A y B cada vez que la señal de reloj hace una transición $0 \rightarrow 1$. Si A y B son altas en esta transición, se introduce por corrimiento un 1 lógico en el dispositivo. En caso contrario,



Entradas				Salidas			
$\overline{\text{Clear}}$	Reloj	A	B	Q_A	Q_B	...	Q_H
L	×	×	×	L	L		L
H	L	×	×	Q_{A0}	Q_{B0}		Q_{H0}
H	↑	H	H	H	Q_{An}		Q_{Gn}
H	↑	L	×	L	Q_{An}		Q_{Gn}
H	↑	×	L	L	Q_{An}		Q_{Gn}

Q_{A0}, Q_{B0}, Q_{H0} = niveles de Q_A, Q_B, Q_H, respectivamente, antes de establecer las condiciones estacionarias indicadas de las entradas.

Q_{An}, Q_{Gn} = niveles de Q_A, Q_G, respectivamente, antes de la transición ↑ más reciente del reloj (corrimiento de 1 bit).

(c)

Figura 7.4 Registro de corrimiento SN74164 entrada serial, salida serial/en paralelo. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos. **(c)** Tabla funcional. **(d)** *Pines* del paquete. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

se introduce un 0 lógico. Observe que se introduce un 1 o un 0 en cada transición $0 \rightarrow 1$ del reloj. Por último, otra señal $\overline{Clear}$ lleva todas las salidas del *flip-flop* a 0. Se genera un patrón interesante de señales de retardo en las salidas de los

flip-flops de este dispositivo. Cada patrón se retarda con respecto de su predecesor un intervalo de pulso de reloj (el tiempo entre las transiciones 0 → 1 en el reloj).

La figura 7.4c presenta la tabla funcional. Examine la tabla e interprete sus filas y columnas de la manera siguiente. Una señal baja en la línea $\overline{Clear}$ mantiene las ocho salidas en 0 lógico. La línea $\overline{Clear}$ debe mantenerse alta para que opere el registro de corrimiento. Con la línea $\overline{Clear}$ alta y la línea *Clock* baja, los ocho *flip-flops* del dispositivo mantienen sus valores iniciales (Q_{i0}, para $i = A,..., H$). En este estado, el dispositivo busca transiciones en sentido positivo (0→1) en el reloj para recorrer los datos hacia abajo en la cadena de *flip-flops*. Si ocurre una transición positiva (una transición 0 → 1, denotada mediante una flecha hacia arriba en la tabla) en el instante t_{n+1}, las salidas de los *flip-flops* Q_B, Q_C,...,Q_H asumirán los valores de sus *flip-flops* vecinos del lado izquierdo en el instante t_n. Por ejemplo, Q_B en el instante t_{n+1} es igual a Q_A en el instante t_n, o $Q_B = Q_{An}$. El valor de la salida Q_A en el instante t_{n+1} queda determinado por las entradas A y B. Si A y B son altas, entonces Q_A es alta. En caso contrario, Q_A es baja.

Por último, las conexiones de *pines* del SN74164 se muestran en la figura 7.4d. Observe que el dispositivo utiliza los 14 *pines*. También hemos insertado los números de los *pines* en la figura 7.4d en el diagrama lógico de la figura 7.4a para conveniencia del usuario del dispositivo.

SN7496

Éste es un registro de corrimiento de 5 bits, de entrada serial, salida serial y paralelo, con *clear* asíncrono común y *preset* asíncrono. Ver el diagrama lógico de la figura 7.5a. SN7496 sólo tiene una línea *de entrada serial*. Tiene las demás características del SN74164 más una característica adicional: *preset* asíncrono. Observe que cada *flip-flop* SR tiene una entrada *Preset i* con compuertas ($i = A,..., E$). Se realiza un AND entre la línea común *Preset enable* y cada señal *Preset i* para enviarla a cada *flip-flop*. Podemos utilizar esta característica junto con la línea $\overline{Clear}$ para cargar con datos el registro de corrimiento de manera paralela. Primero aplicamos un pulso $\overline{Clear}$ bajo activo (1 → 0 → 1) al dispositivo para llevar los cinco *flip-flops* al estado 0 lógico. Después, aplicamos a los *flip-flops* las señales individuales *Preset i* con los datos por transferir al registro mediante un pulso *Preset enable* alto activo (0 → 1 → 0). Observe que las salidas del *flip-flop* se mantienen con sus valores en las líneas *Preset i* si la línea *Preset enable* se mantiene alta. Por tanto, el pulso *preset enable* debe regresar al valor 0 lógico antes de aplicar los pulsos de control de corrimiento a la línea *clock*.

La figura 7.5b muestra algunas operaciones típicas del SN7496. Primero se muestra la característica de entrada serial, salida serial. Se aplica un pulso $\overline{Clear}$ bajo activo para que todos los *flip-flops* pasen al estado 0 lógico. Después se aplica una señal alta activa a la línea *De entrada serial*, que entra al *flip-flop A* en la primera transición 0 → 1 del reloj. Después se elimina la señal de entrada serial para que cada transición 0 → 1 posterior del reloj recorra el 1 lógico hacia abajo en la cadena de cinco *flip-flops*. Después del quinto pulso de reloj, los cinco *flip-flops* han regresado al estado nulo. A continuación daremos un ejemplo de carga paralela del registro. Un pulso *Preset enable* alto activo carga el registro con datos binarios, $(Q_E Q_D Q_C Q_B Q_A) = (01011)$. Por último, los datos introducidos

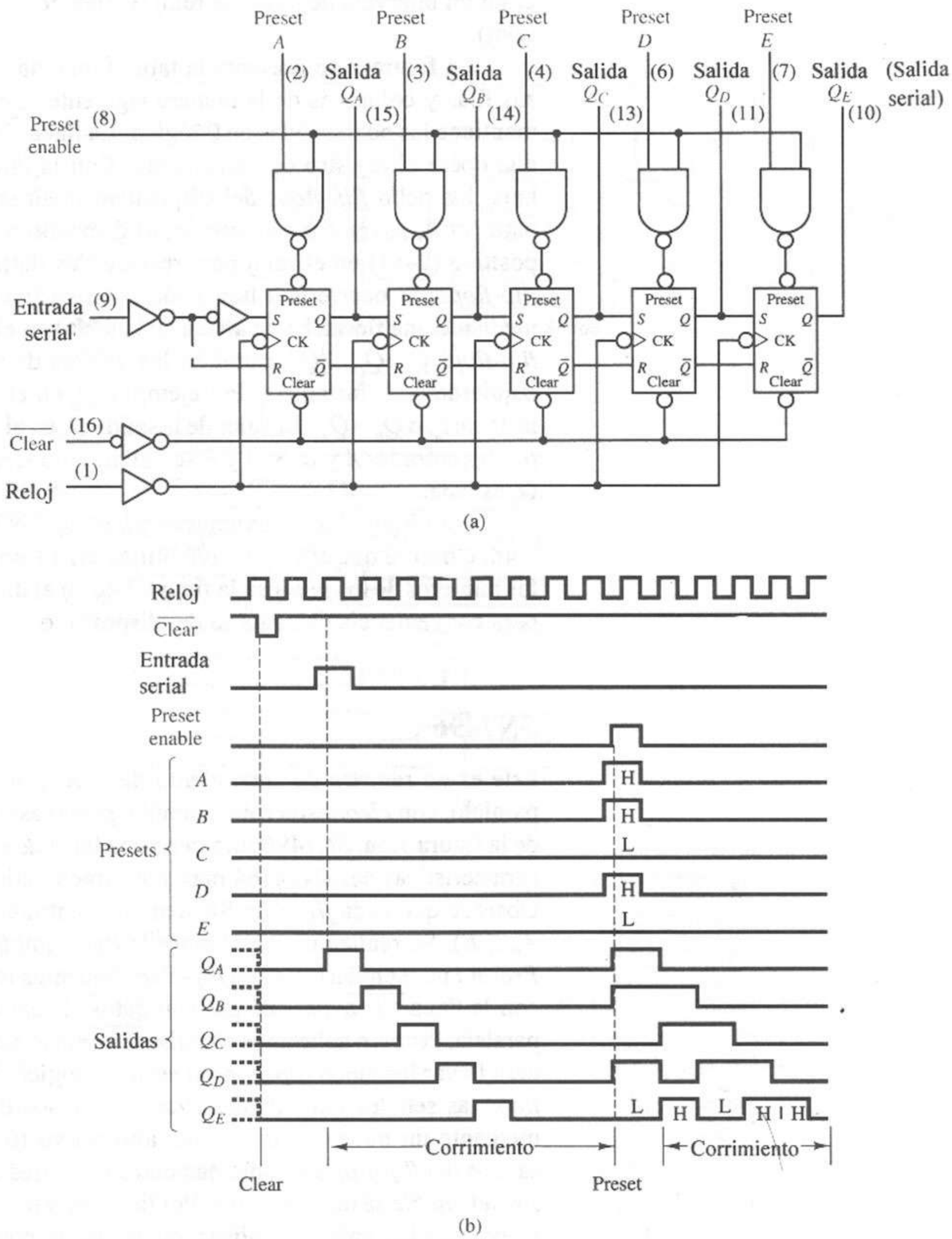


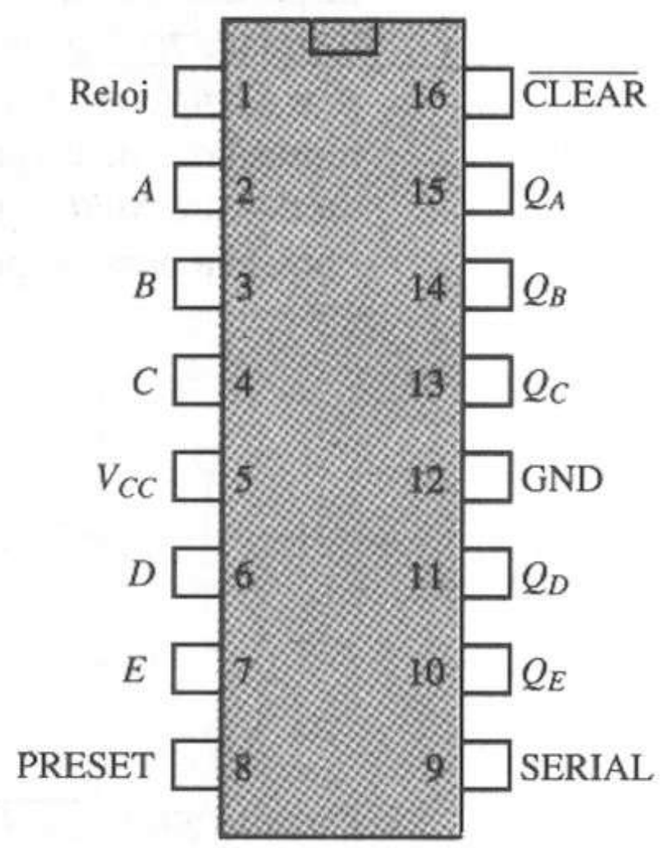
Figura 7.5 Registro de corrimiento SN7496 de 5 bits, entrada serial, salida serial/en paralelo. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

al registro se convierten de forma paralela a serial aplicando cuatro pulsos adicionales del reloj para sacar los cinco bits del *pin* de salida Q_E.

Entradas									Salidas				
	Preset	Preset											
$\overline{\text{Clear}}$	Enable	A	B	C	D	E	Reloj	Serial	Q_A	Q_B	Q_C	Q_D	Q_E
L	L	×	×	×	×	×	×	×	L	L	L	L	L
L	×	L	L	L	L	L	×	×	L	L	L	L	L
H	H	H	H	H	H	H	×	×	H	H	H	H	H
H	H	L	L	L	L	L	L	×	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}	Q_{E0}
H	H	H	L	H	L	H	L	×	H	Q_{B0}	H	Q_{D0}	H
H	L	×	×	×	×	×	L	×	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}	Q_{E0}
H	L	×	×	×	×	×	↑	H	H	Q_{An}	Q_{Bn}	Q_{Cn}	Q_{Dn}
H	L	×	×	×	×	×	↑	L	L	Q_{An}	Q_{Bn}	Q_{Cn}	Q_{Dn}

Q_{A0}, Q_{B0}, etc. = niveles de Q_A, Q_B, etc., respectivamente, antes de establecer las condiciones estacionarias indicadas de las entradas. Q_{An}, Q_{Bn}, etc. = niveles de Q_A, Q_B, respectivamente, antes de la transición ↑ más reciente del reloj.

(c)



(d)

Figura 7.5 (continuación) Registro de corrimiento SN7496 de 5 bits, entrada serial, salida serial/en paralelo. **(c)** Tabla funcional. **(d)** *Pines* del paquete (vista superior). *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

La figura 7.5c muestra la tabla funcional del SN7496. Como la señal $\overline{Clear}$ es baja activa y *Preset enable* es alta activa, no es usual aplicar esta combinación al dispositivo. Si aplicamos ($\overline{Clear}$, *Preset enable*) = (0,1), los *flip-flops* SR asumirán el estado nulo, como vemos en la fila 2 de la tabla. Consideremos la fila 1 de la tabla: ($\overline{Clear}$, *Preset enable*) = (0,0). Ésta es la función *Clear*. Las filas 3, 4 y 5 muestran varios ejemplos de preestablecimiento. En la fila 3, todas las señales *Preset i* individuales son altas, de modo que todas las salidas de *flip-flop* Q_i se hacen altas. En las filas 4 y 5, las señales *Preset i* seleccionadas son bajas, lo que deja a las Q_i de esos *flip-flops* Q_i sin modificación por la acción de

la señal *Preset enable*. Las últimas tres filas de la tabla muestran las operaciones de corrimiento. La fila 6 muestra que no hay corrimientos mientras la señal de reloj es baja. Las filas 7 y 8 muestran que *Serial input* entra al registro y todos los bits de datos se recorren una posición a la derecha cuando sucede cada transición de reloj $0 \rightarrow 1$.

La figura 7.5d muestra la configuración de los *pines* de este paquete dual en línea de 16 *pines*.

SN74165

Este dispositivo es un registro de corrimiento de 8 bits, de entrada serial, salida serial, con características de carga asíncrona e inhibición del reloj. Analice el diagrama lógico de la figura 7.6a. Observe que este circuito tiene 12 entradas y sólo 2 salidas. Para explicar su función, hemos ampliado la primera celda del registro en la figura 7.6b. Observe primero que no existe una señal *Clear*. El símbolo lógico para cada *flip-flop* indica que éste opera como un *flip-flop* D disparado por transición positiva con terminales *preset* (rotulada *S*) y *clear* (rotulada *R*). Las terminales *S* y *R* del *flip-flop* están controladas por las entradas *A* y *Shift*/$\overline{load}$. *Shift* representa el funcionamiento de "corrimiento" y $\overline{Load}$ representa "no carga". En otras palabras, *Shift* y *Not-load* son la misma señal, de modo que *Shift* y *Load* son señales complementarias. Utilizamos el álgebra booleana para despejar *S* y *R* en las ecuaciones lógicas y determinar que

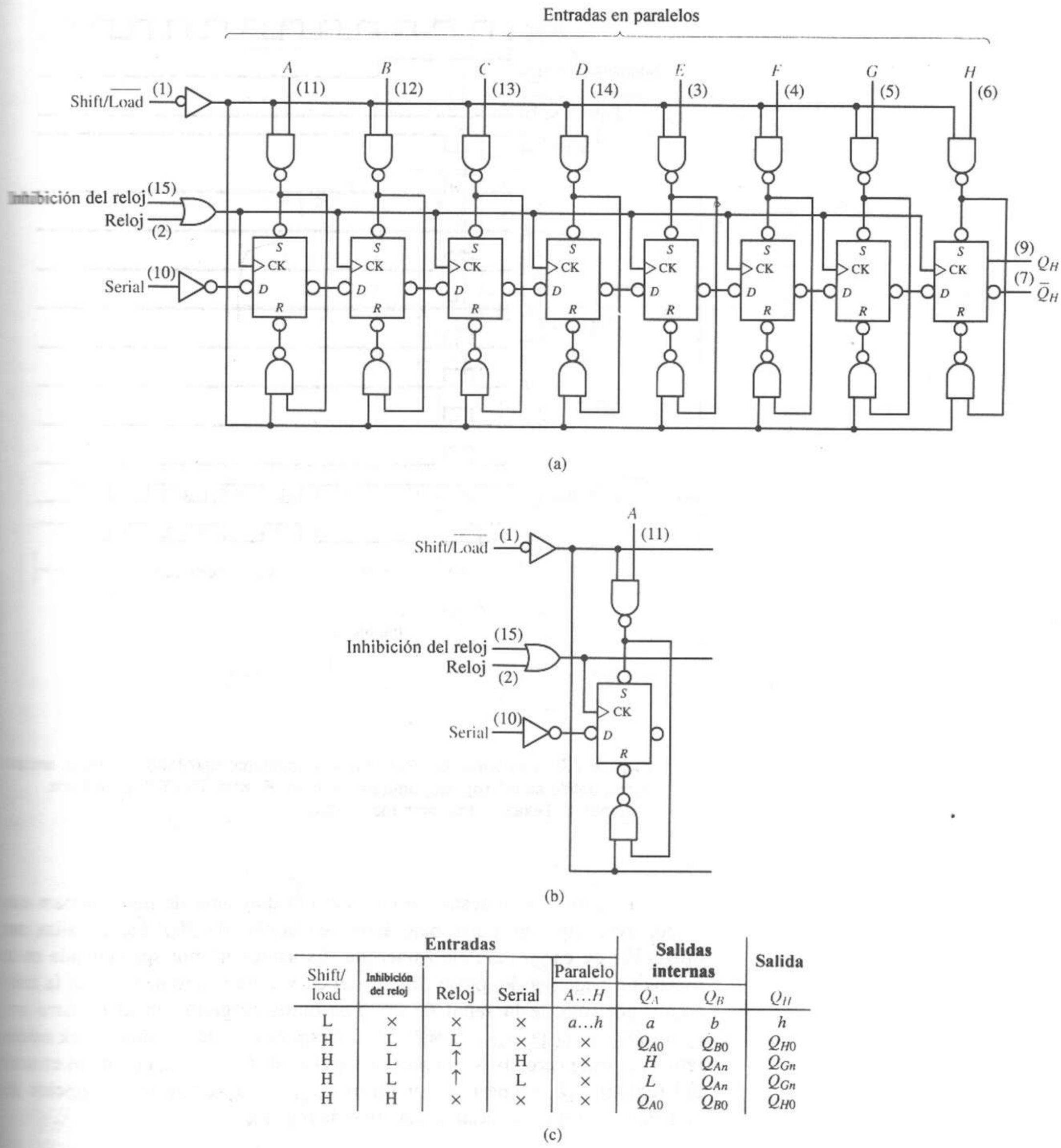
$$S = A\,\overline{(Shift/\overline{Load})} \tag{7.1}$$

$$\begin{aligned} R &= \overline{S}\,\overline{(Shift/\overline{Load})} \\ &= \overline{[A\,\overline{(Shift/\overline{Load})}]}\;\overline{(Shift/\overline{Load})} \\ &= [\bar{A} + (Shift/\overline{Load})]\overline{(Shift/\overline{Load})} \\ &= \bar{A}\,\overline{(Shift/\overline{Load})} \end{aligned} \tag{7.2}$$

Cuando *Shift*/$\overline{Load}$ es alta, las dos entradas del *flip-flop S* y *R* son bajas, de modo que el *flip-flop* opera en su modo de reloj. Pero cuando *Shift*/$\overline{Load}$ es baja, entonces $S = A$ y $R = \bar{A}$, de modo que la salida del *flip-flop* asumirá el valor de la entrada *A*; éste es el modo de operación de *carga paralela* del dispositivo.

Ahora examine las señales *Clock* y *Clock inhibit*. Cuando *Clock inhibit* es alta, la salida de la compuerta OR se mantiene alta y evita (inhibe) que cualquier transición $0 \rightarrow 1$ de la entrada de reloj (*Clock*) alcance la terminal de entrada de reloj del *flip-flop*, *C*1. Así, *Clock inhibit* = *1* es el modo de operación de *conservación de datos* para este dispositivo.

El funcionamiento del SN74165 se resume en la tabla funcional de la figura 7.6c. La fila 1 de la tabla describe el modo de operación de carga paralela ya analizado. La última línea en la tabla describe el modo de retención, en el que la señal *Clock inhibit* bloquea cualquier actividad sobre la entrada de reloj. Las otras tres líneas (con *Clock inhibit* baja) describen los modos de funcionamiento de corrimiento. La segunda línea de la tabla ilustra el dispositivo y su estado inicial con la entrada de reloj baja en espera de una transición. Todos los *flip-flops* tienen una condición estable. En la transición positiva del pulso de reloj, la tercera y cuarta filas de la tabla indican que los datos de entrada seriales entran en la primera celda, mientras que las celdas restantes recorren sus datos a la derecha.



Entradas					Salidas internas		Salida
Shift/load	Inhibición del reloj	Reloj	Serial	Paralelo $A \ldots H$	Q_A	Q_B	Q_H
L	×	×	×	$a \ldots h$	a	b	h
H	L	L	×	×	Q_{A0}	Q_{B0}	Q_{H0}
H	L	↑	H	×	H	Q_{An}	Q_{Gn}
H	L	↑	L	×	L	Q_{An}	Q_{Gn}
H	H	×	×	×	Q_{A0}	Q_{B0}	Q_{H0}

(c)

Figura 7.6 Registro de corrimiento SN74165 de 8 bits, entrada serial, salida serial. **(a)** Diagrama lógico. **(b)** Celda A. **(c)** Tabla funcional.

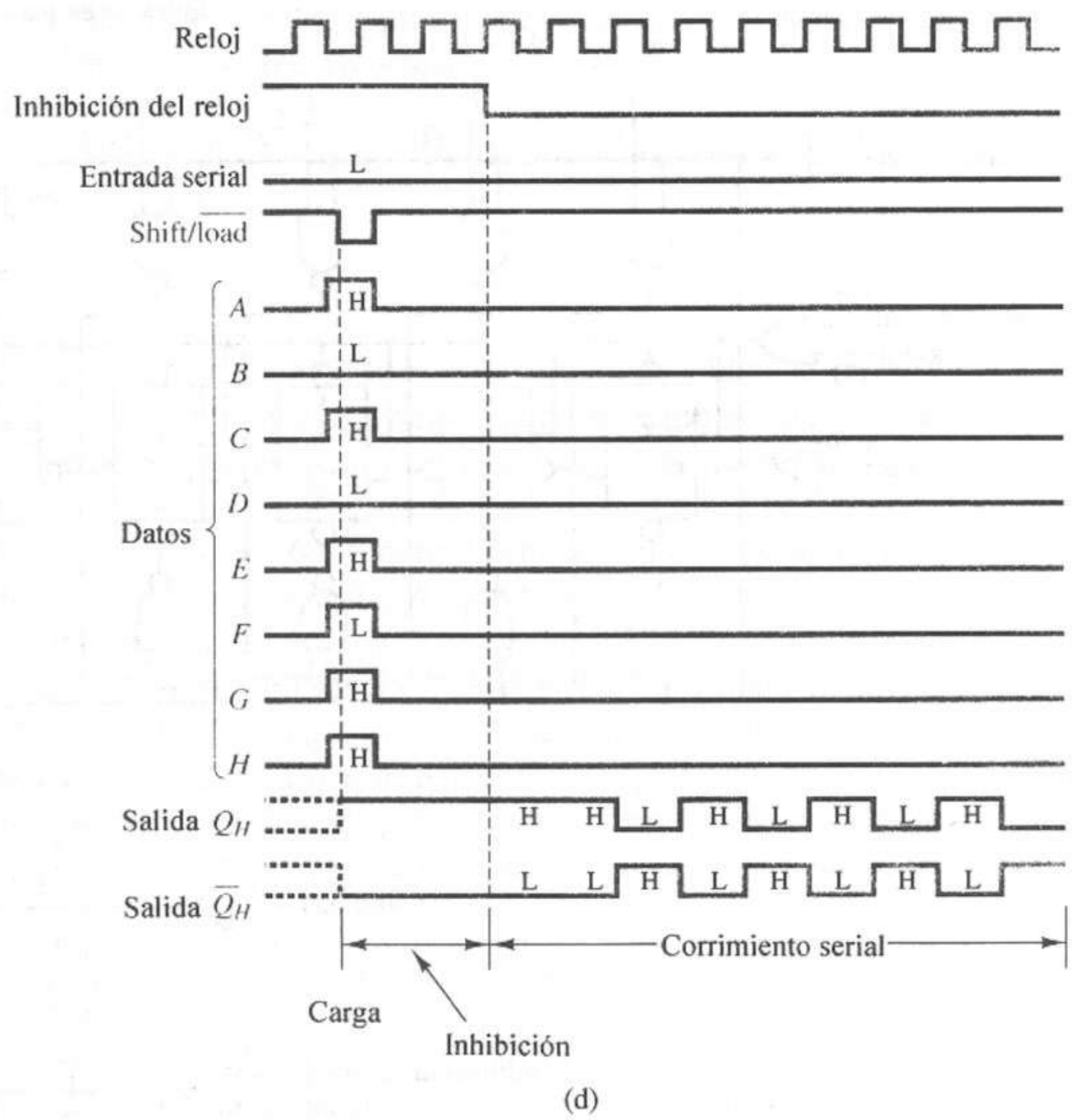
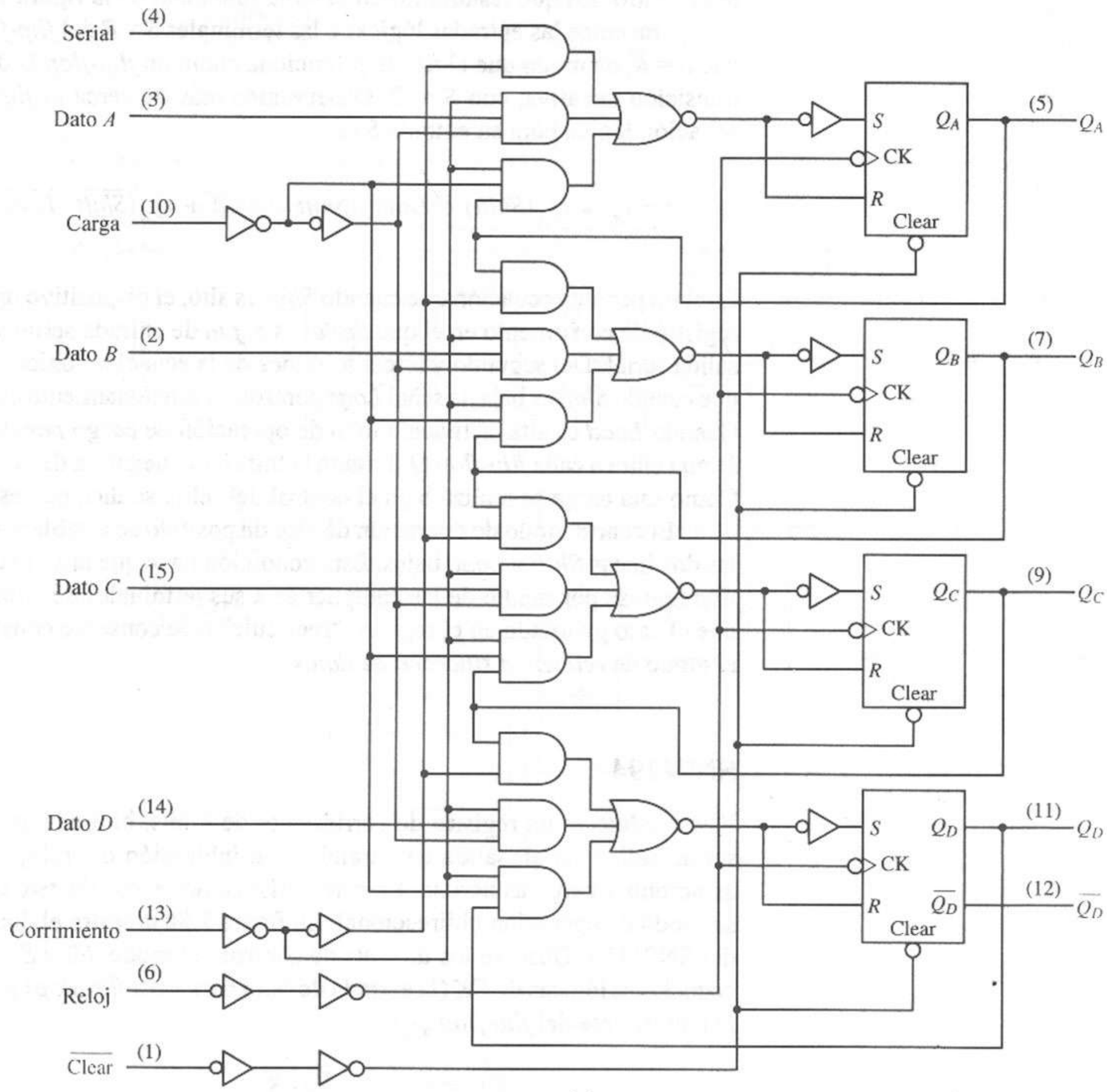


Figura 7.6 (continuación) Registro de corrimiento SN74165, de 8 bits, entrada serial, salida serial. **(d)** Diagrama de tiempos. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

La figura 7.6d muestra un ejemplo del diagrama de tiempos para este dispositivo. Aquí, un pulso bajo activo en la entrada *Shift/Load* realiza una operación de carga paralela asíncrona. Es asíncrona por ser activada en la transición negativa del pulso *Shift/Load* en vez de sincronizarse con la transición positiva de la señal *Clock*. Los datos cargados en el registro son $(Q_H Q_G Q_F Q_E Q_D Q_C Q_B Q_A) = (11010101)$. Después de que la señal *Clock inhibit* retoma el valor cero, los siguientes siete pulsos de *Clock* sacan los datos en serie del registro (Q_H primero y por último Q_A). Esto demuestra la función de conversión paralela a serial de este tipo de registro.

SN74179

Éste es un módulo de registro de corrimiento de 4 bits, de entrada serial, salida serial, salida en paralelo, con *Clear* común asíncrono, carga síncrona y retención de datos síncrona. Examine el diagrama lógico de la figura 7.7a. Los *flip-flops* SR tienen una terminal *Clear* pero no una función *preset*. La línea de control



(a)

Entradas			Modo
$\overline{\text{Clear}}$	Corrimiento	Carga	
L	×	×	*Clear* asíncrono
H	H	×	Corrimiento
H	L	H	Carga síncrona
H	L	L	Retención de datos

(b)

Figura 7.7 Registro de corrimiento SN74179, de 4 bits, entrada serial, salida serial/en paralelo. **(a)** Diagrama lógico. **(b)** Tabla funcional. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

$\overline{Clear}$ común asíncrona hace que todos los *flip-flops* del registro estén en el estado 0 lógico cuando son bajos. Cuando $\overline{Clear}$ es alto, existen otros tres modos del registro, los que resumimos en la tabla funcional de la figura 7.7b.

Tracemos las entradas lógicas a las terminales S y R del *flip-flop*. Observe que $S = \overline{R}$, de modo que el *flip-flop* funciona como un *flip-flop* D disparado por transición negativa, con $S = D$. Observemos más de cerca el *flip-flop* Q_B. La ecuación lógica para su entrada S es

$$S_B = Q_A(Shift) + Data_B(\overline{Shift} \cdot Load) + Q_B(\overline{Shift} \cdot \overline{Load}) \qquad (7.3)$$

Es claro por esta ecuación que cuando *Shift* es alto, el dispositivo opera como un registro de corrimiento en el que *Serial* es el *pin* de entrada serial y Q_D el *pin* de salida serial. Del segundo y tercer términos de la ecuación lógica, podemos ver que cuando *Shift* es baja, la señal *Load* controla el funcionamiento del dispositivo. Cuando *Load* es alta, activa el modo de operación de *carga paralela síncrona*: *Data i* entra a cada *flip-flop* Q_i durante la transición negativa de la señal de reloj. Como esta carga se realiza bajo el control del reloj, se dice que es síncrona.

El cuarto modo de operación de este dispositivo se establece manteniendo las dos líneas *Shift* y *Load* bajas. Esta condición hace que la salida de cada *flip-flop* regrese por medio de las compuertas a sus terminales de entrada de modo que el dato guardado en el registro "recircule" o se conserve constante. Éste es el modo de *retención síncrona de datos*.

SN74194

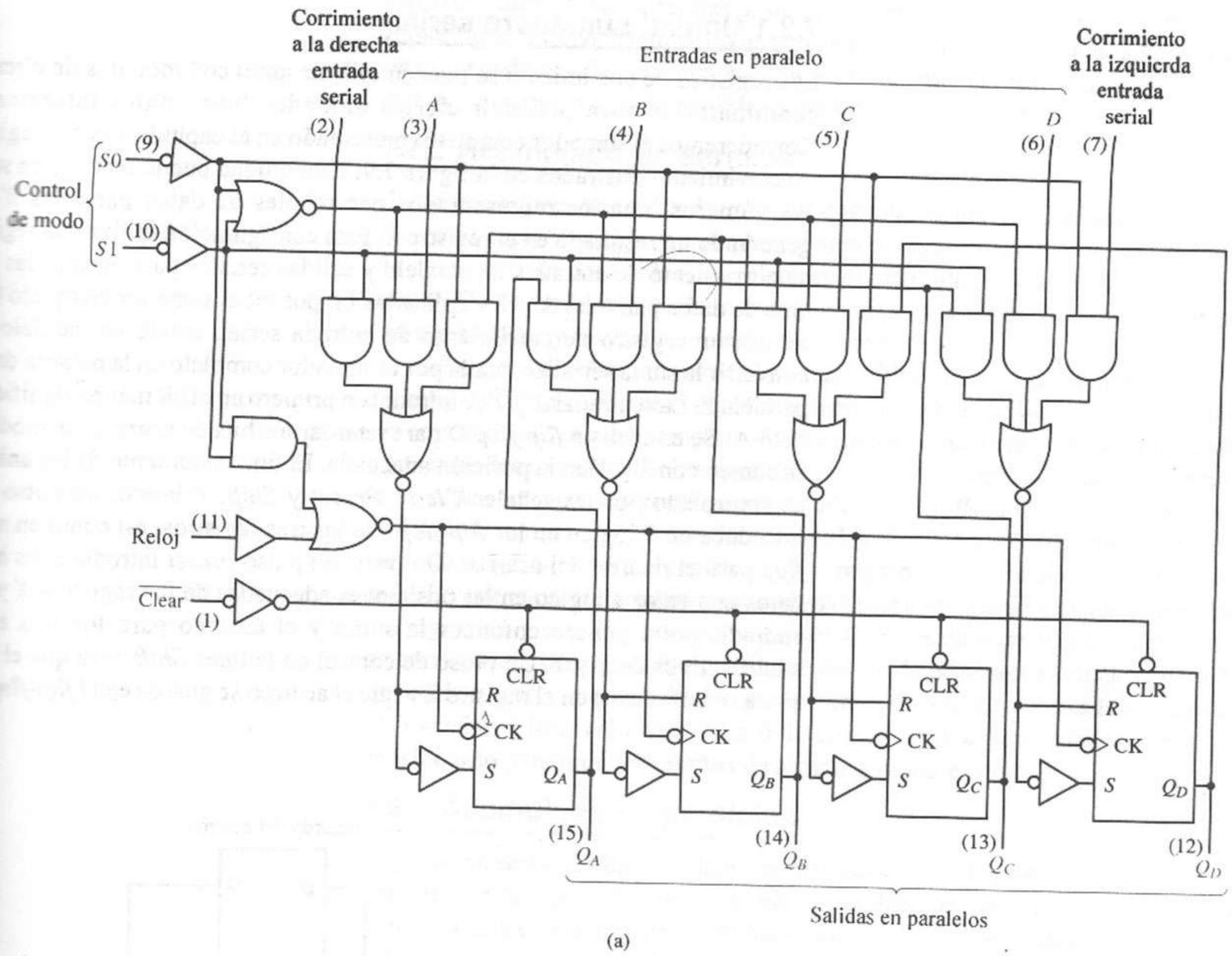
Este módulo es un registro de corrimiento de 4 bits, bidireccional, de entrada serial, salida serial, salida en paralelo, con inhibición del reloj, *Clear* común asíncrono y carga asíncrona. La característica novedosa de este dispositivo es su modo de operación bidireccional. La figura 7.8a muestra el diagrama lógico del SN74194. Observe los dos bits de control del modo, $S0$ y $S1$. Examine las ecuaciones lógicas de CK (la entrada de reloj para cada *flip-flop*) y S_B (la entrada D equivalente del *flip-flop* Q_B):

$$CK = Clock + \overline{S0} \cdot \overline{S1} \qquad (7.4)$$

$$S_B = Q_C \cdot \overline{S0} + Q_A \cdot \overline{S1} + B \cdot S0 \cdot S1 \qquad (7.5)$$

Por la ecuación 7.4 para CK, vemos que cuando $S0$ y $S1$ son ambas bajas, el reloj se inhibe, debido a que su actividad es encubierta por un 1 lógico constante producido por el segundo término de la ecuación. Así, para los modos de operación con reloj se requiere que $S0$ o $S1$ sean altos.

Examinemos los modos de funcionamiento con reloj. Por la ecuación 7.5, si $S1$ es alta y $S0$ es baja, entonces $S_B = Q_C$; es decir, la salida del *flip-flop* Q_C se aplica a las terminales de entrada del *flip-flop B*; ésta es el funcionamiento de *corrimiento a la izquierda*. Al intercambiar los valores de los bits de control del modo ($S0$ alto y $S1$ bajo), la ecuación 7.5 se reduce a $S_B = Q_A$, aplicando los datos de Q_A al *flip-flop B*; éste es el modo de operación de *corrimiento a la derecha*. La cuarta condición de los bits de control del modo ($S0$ y $S1$ altos) envía el bit de entrada de datos B paralelo externo al módulo a las entradas S y R del *flip-flop B*; es decir, la ecuación 7.5 se reduce a $S_B = B$; éste es el modo de



(a)

Entradas				Señales internas ($i = A, B, C, D$)	
$\overline{\text{Clear}}$	$S0$	$S1$	Reloj	S_i	Modo
L	×	×	×	×	*Clear* asíncrono
H	L	L	×	×	Inhibición del reloj (retención de datos)
H	L	H	↑	Q_{i+1}	Corrimiento a la izquierda
H	H	L	↑	Q_{i-1}	Corrimiento a la derecha
H	H	H	↑	i	Carga en paralelo

(b)

Figura 7.8 Registro de corrimiento SN74194, bidireccional, entrada serial, salida serial/en paralelo. **(a)** Diagrama lógico. **(b)** Tabla funcional. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

operación *síncrono de carga paralela*. La tabla funcional de la figura 7.8b resume los cuatro modos de funcionamiento.

El SN74194 es un dispositivo de 16 *pines*.

7.2 Ejemplos de diseño con registros

7.2.1 Unidad sumadora serial

Los registros de corrimiento se pueden utilizar junto con módulos de circuitos combinatorios para producir ciertas unidades funcionales interesantes. Consideremos el sumador completo (presentado en el capítulo 4) y los registros de corrimiento ilustrados en la figura 7.9. Esta unidad puede servir para sumar los números binarios representados por señales de datos paralelas X y Y, generando un resultado en el registro Z. Esta configuración utiliza dos registros de corrimiento de entradas en paralelo y salidas seriales para aceptar las palabras de datos paralelas X y Y y aplicarlas bit por bit al sumador completo (FA). Se usa un registro de corrimiento de entrada serial, salida en paralelo para convertir la suma serial generada por el sumador completo en la palabra de dato paralela Z. Las entradas X y Y se introducen primero en el bit menos significativo del FA. Se emplea un *flip-flop* D para retardar los bits de acarreo, de modo que se sumen con X y Y en la posición adecuada. El funcionamiento de las unidades es controlado por las señales *Clear*, *Preset* y *Shift*. Primero, un pulso *clear* introduce un 0 lógico en los *flip-flops* de los tres registros, así como en el *flip-flop* para el retardo del acarreo. Después, un pulso *preset* introduce los bits de datos con valor 1 lógico en las posiciones adecuadas de los registros X y Y. El sumador total genera entonces la suma y el acarreo para los bits menos significativos de X y Y. Un pulso de control en la línea *Shift* hace que el bit de suma se introduzca en el registro Z y que el acarreo se guarde en el *flip-flop* para

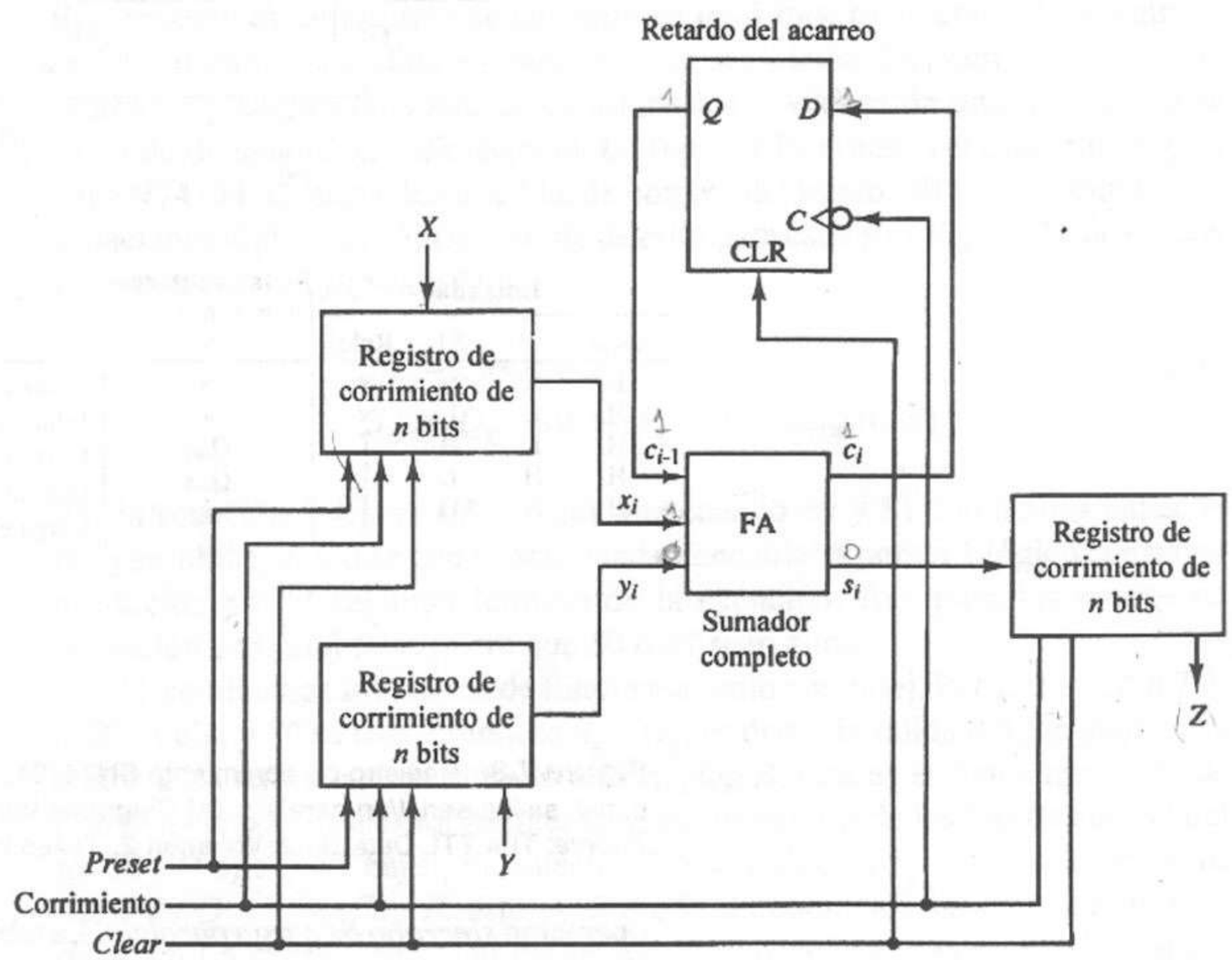


Figura 7.9 Unidad sumadora serial.

el retardo del acarreo. El sumador completo genera de nuevo las salidas de la suma y el acarreo para los siguientes bits más significativos de X y Y. Con cada pulso de corrimiento, se realiza la suma de otra posición de bit. Este proceso se repite hasta que el n-ésimo pulso de corrimiento introduce el bit más significativo de la suma de X y Y en el registro Z. En ese momento, el funcionamiento termina y la suma paralela está disponible en las salidas del registro Z.

7.2.2 Acumuladores seriales

Un *acumulador* es una unidad sumadora que totaliza una serie de datos binarios. Funciona como una caja registradora donde podemos sumar cualquier número de elementos de datos binarios mientras el acumulador guarda la suma total actual. Como en la caja registradora, podemos exceder el intervalo operativo del acumulador y producir un desbordamiento.

Podemos diseñar un acumulador serial utilizando la unidad sumadora serial de la figura 7.9, eliminando el registro Y y permitiendo al registro Z retroalimentar las entradas del sumador completo. Ver la figura 7.10a. Para un funcionamiento adecuado, debemos utilizar la siguiente secuencia de control. Primero, pulsamos la línea de control *Clear* para inicializar todos los *flip-flops* en 0 lógico. Después aplicamos un pulso *Preset*, seguido de n pulsos *Shift*. Ahora, el primer elemento de datos binarios está en el registro Z, y el registro X está de nuevo en el estado nulo. En este punto, se proporcionan nuevos datos a las líneas de entrada paralelas del registro X y se aplica otro pulso *preset*. Después de otras n señales de pulso de corrimiento, la suma de los dos elementos de datos aparecen en el registro Z. Se puede sumar cualquier número de elementos de datos, mientras la suma total no exceda el intervalo de n bits del registro Z. Un acumulador serial es satisfactorio para muchos diseños de sistemas digitales de baja velocidad.

7.2.3 Acumuladores paralelos

Para operaciones de alta velocidad, es más adecuado el diseño del acumulador paralelo de la figura 7.10b. Este diseño se basa en una unidad sumadora seudoparalela con un registro de almacenamiento Z con retroalimentación. Para el funcionamiento adecuado de este circuito se requiere que se pulse el control *Clear* para inicializar el circuito. Después es necesario un pulso *Accumulate* para sumar cada nuevo elemento de datos. Esta unidad es más rápida que el diseño serial y su funcionamiento es mucho menos complejo. Los módulos acumuladores se usan mucho en las aplicaciones del diseño lógico digital.

7.3 Contadores

Los *contadores* son una clase de circuitos lógicos secuenciales que llevan la cuenta de una serie de pulsos de entrada; los pulsos de entrada pueden ser regulares o irregulares. El contador es parte fundamental de muchas aplicaciones lógicas digitales. Se utiliza en unidades de control de tiempos, circuitos de control, generadores de señal y muchos otros dispositivos.

Podemos clasificar los contadores como binarios/no binarios y asíncronos/síncronos. A continuación describimos varios ejemplos de diseños de contadores. En este capítulo, utilizaremos como ejemplo algunos dispositivos contadores de la serie SN7400 para ilustrar la amplia gama de opciones disponibles para el diseñador lógico. Analizaremos los diseños síncronos y asíncronos, y características tales como el *clear* síncrono y asíncrono, la activación, la carga

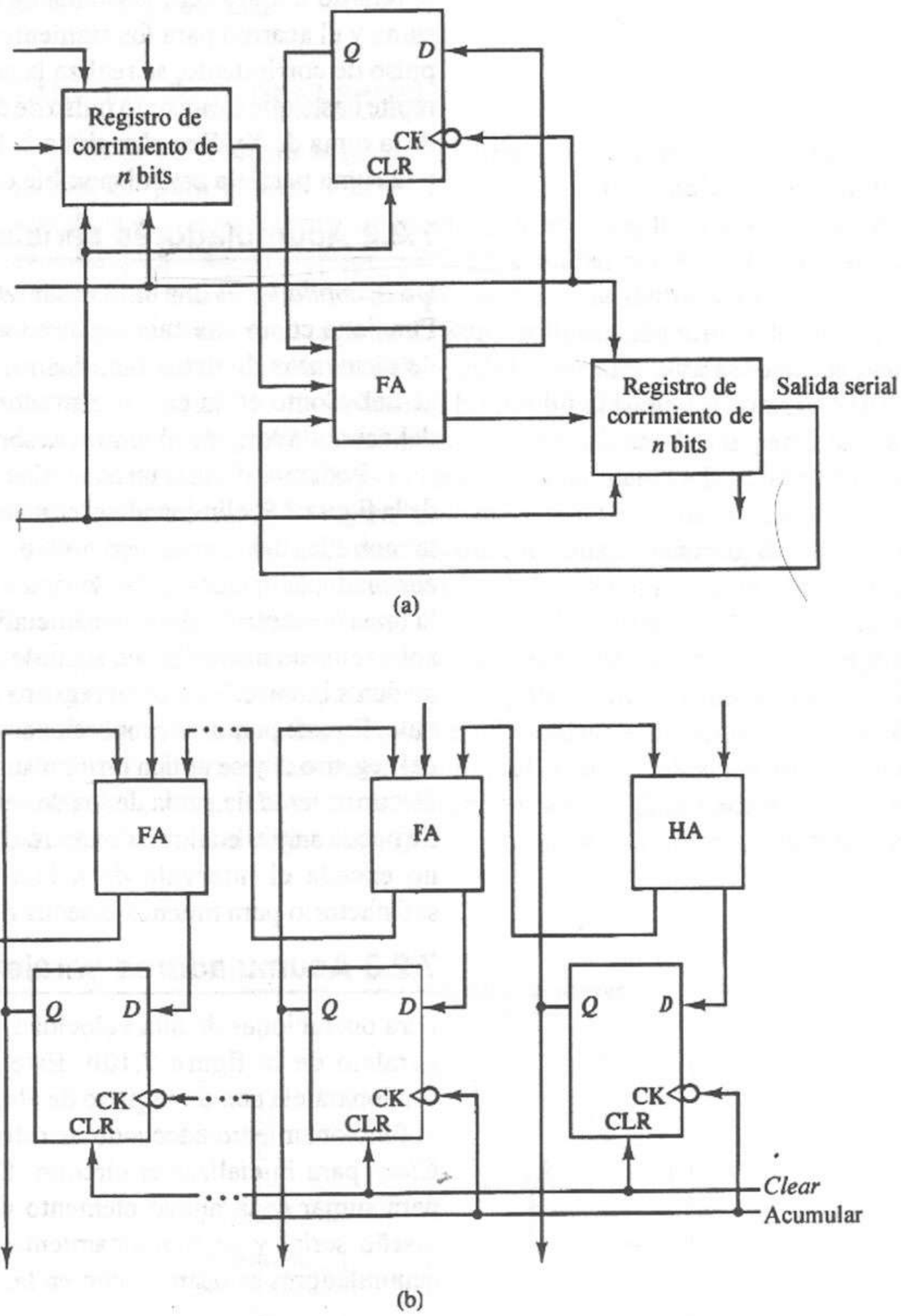


Figura 7.10 Acumuladores. **(a)** Unidad serial. **(b)** Unidad en paralelo.

síncrona y asíncrona y la salida con acarreo en cascada. Ilustraremos los contadores binarios y de módulo N. La tabla 7.2 resume los contadores que analizaremos. Definiremos las características de cada dispositivo conforme los presentemos.

TABLA 7.2 CONTADORES DE LA SERIE SN7400

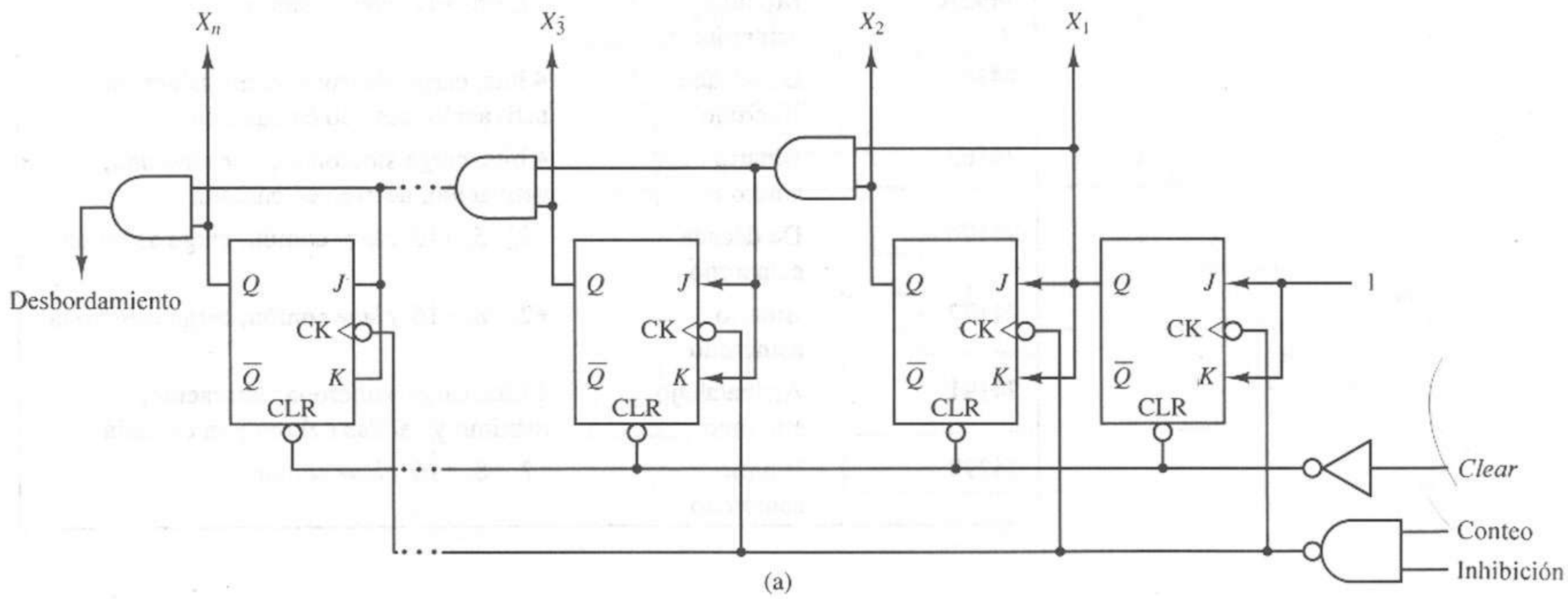
Dispositivo	Tipo	Características
7492A	Módulo asíncrono 12	÷2, ÷6, ÷12, *clear* común
74160	De década síncrono	4 bits, carga síncrona, *clear* asíncrono, activación, acarreo en cascada
74163	Binario síncrono	4 bits, carga síncrona, *clear* síncrono, activación, acarreo en cascada
74176	De década asíncrono	÷2, ÷5, ÷10, *clear* común, carga asíncrona
74177	Binario asíncrono	÷2, ÷8, ÷16, *clear* común, carga asíncrona
74191	Arriba/abajo síncrono	4 bits, carga asíncrona, activación, máximo y salidas del reloj en cascada
74293	Binario asíncrono	÷2, ÷8, ÷16, *clear* común

7.3.1 Contadores binarios síncronos

La figura 7.11a muestra un contador binario síncrono de n bits construido a base de *flip-flops* JK con reloj. Un contador binario de n *flip-flops* debe partir del estado nulo y seguir la secuencia numérica 0, 1, 2, 3,..., 2^n-1, 0, 1, 2,..., etcétera. En otras palabras, el contador tendrá 2^n estados únicos, como se muestra en la figura 7.11b, y repetirá los estados mientras se apliquen pulsos de reloj. La secuencia de estados de la figura 7.11b sugiere el diseño de la figura 7.11a. Observe que cada bit X_i debe complementarse en el siguiente pulso de conteo si todos los bits X_k para $k = 1,..., i-1$ tienen el valor 1 lógico; el bit X_1 siempre se complementa en cada pulso de conteo. Por tanto, podemos utilizar una compuerta AND de dos entradas en cada *flip-flop* del contador para generar una señal de control de alternancia para el siguiente bit más significativo en la cadena del contador. El *flip-flop* contador y los circuitos de control asociados a éste son una *etapa del contador*.

En condiciones de operación normales, las entradas J y K de cada *flip-flop* deben permanecer estables en 1 o 0 lógico mientras el pulso del contador experimenta sus transiciones $0 \rightarrow 1 \rightarrow 0$. Un 1 lógico en la línea de control *clear* obligará a todas las salidas del contador a asumir un 0 lógico y las mantendrá así hasta que la línea *clear* regrese a 0 lógico (su valor lógico normal). La señal de control *Inhibit* sirve para bloquear los pulsos de conteo y dejar al contador en algún estado no nulo, si se requiere tal comportamiento de conservación de datos para una aplicación en particular.

Cuando los contadores lleguen al estado en que todos los valores son iguales a uno, la señal de *desbordamiento* será alta. En algunos diseños de contadores, esta señal de desbordamiento se utiliza para controlar los módulos contadores en cascada y producir contadores con mayor longitud de palabra. En estos casos, la señal de desbordamiento se llama *acarreo de salida en cascada* (RCO).



(a)

X_n	…	X_3	X_2	X_1	
0	…	0	0	0	
0	…	0	0	1	
0	…	0	1	0	
0	…	0	1	1	
0	…	1	0	0	
		⋮			
1	…	1	1	0	
1	…	1	1	1	← Recicla
0	…	0	0	0	
0	…	0	0	1	
0	…	0	1	0	

(b)

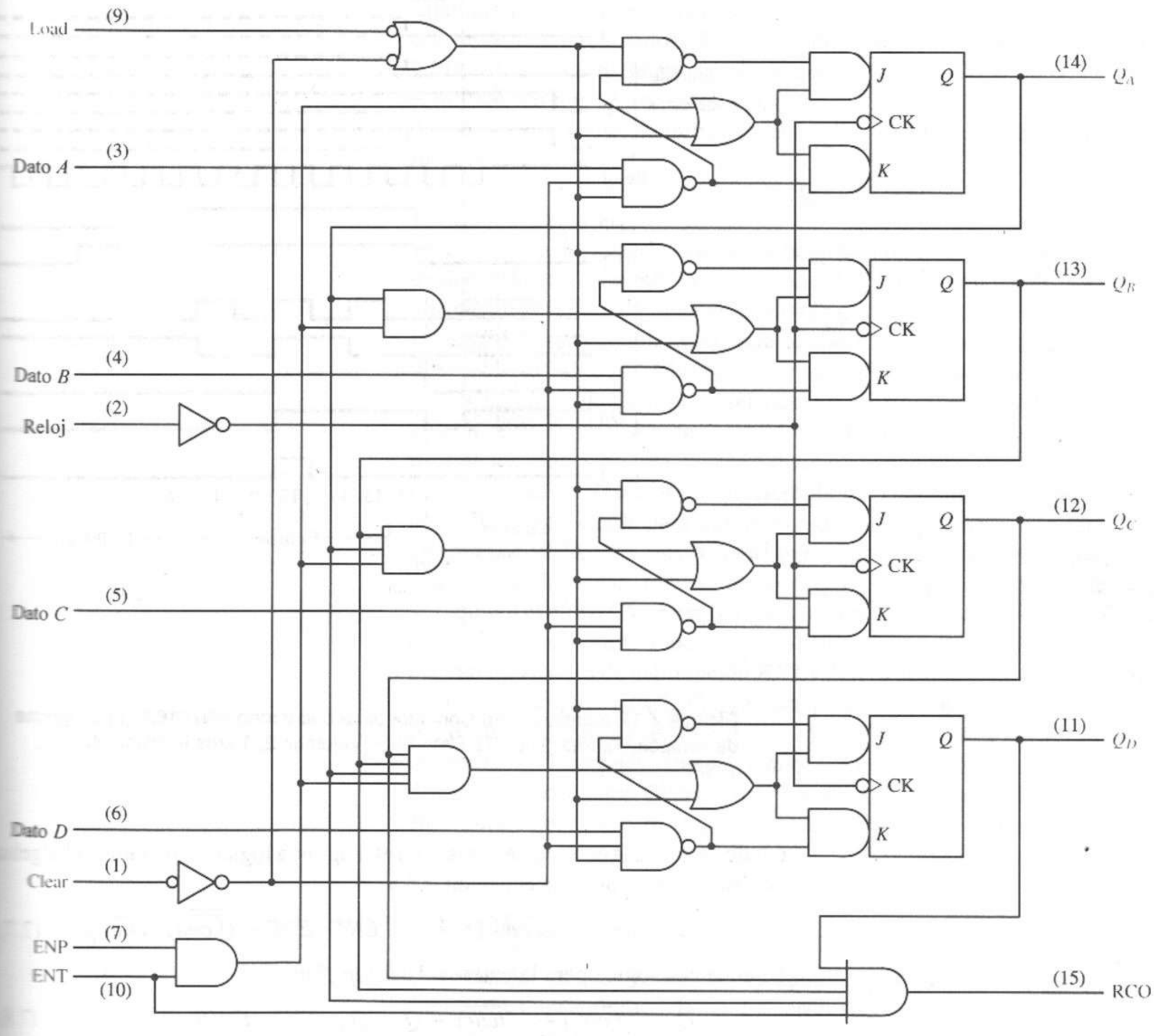
Figura 7.11 Contador binario síncrono. **(a)** Diagrama lógico. **(b)** Secuencia de estados.

SN74163

Éste es un contador síncrono de 4 bits con carga síncrona, *clear* síncrono, activación y acarreo de salida en cascada. La figura 7.12a muestra el diagrama lógico para el dispositivo. Observe que todos los cambios de datos en los *flip-flops* JK suceden en la transición positiva del reloj externo rotulado *Clock*. Por tanto, las operaciones *clear* y *load* se sincronizan con el reloj en este dispositivo.

Examinemos el funcionamiento de una etapa típica del contador, digamos *C*. Podemos escribir la entrada *K* del *flip-flop* como

$$K_C = \frac{[\overline{(\overline{Load})} + \overline{(\overline{Clear})} + Q_B \cdot Q_A \cdot ENT \cdot ENP]}{\cdot[(\overline{(\overline{Load})} + \overline{(\overline{Clear})}) \cdot \overline{Clear} \cdot Data_C]} \tag{7.6}$$



(a)

Entradas				
$\overline{\text{Clear}}$	$\overline{\text{Load}}$	ENT	ENP	Modo
L	×	×	×	*Clear* síncrono
H	L	×	×	Carga síncrona
H	H	H	H	Conteo
H	H	L	×	Retención
H	H	×	L	Retención

(b)

Figura 7.12 Contador binario síncrono SN74163. **(a)** Diagrama lógico. **(b)** Tabla funcional. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

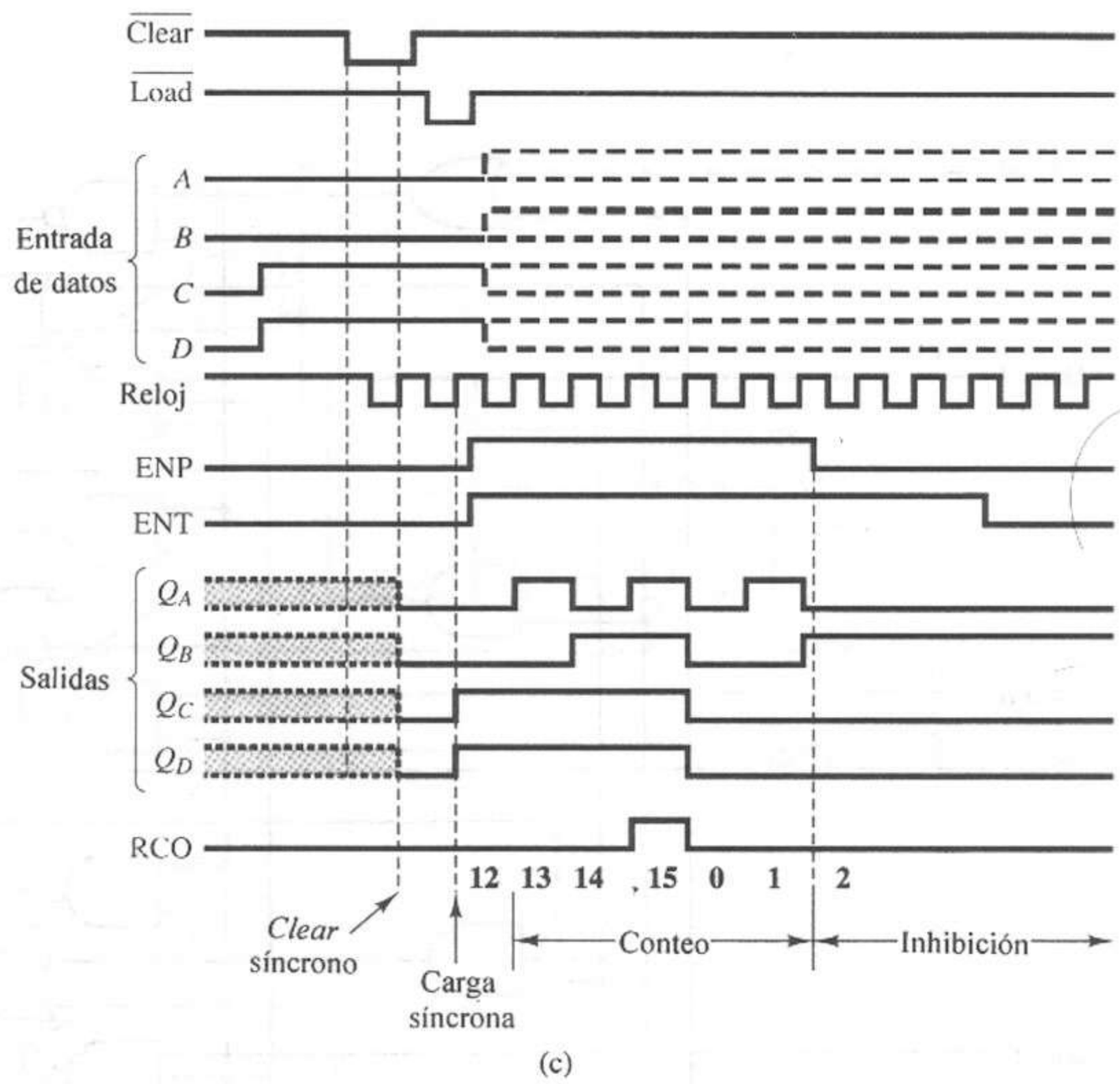


Figura 7.12 (continuación) Contador binario síncrono SN74163. **(c)** Diagrama de tiempos. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

donde $Data_C$ es la entrada de datos C en el diagrama lógico. Utilizamos el álgebra booleana para reducir la ecuación 7.6 a

$$K_C = \overline{(\overline{Clear})} + \overline{Load} \cdot Q_B \cdot Q_A \cdot ENT \cdot ENP + \overline{(\overline{Load})} \cdot \overline{Data_C} \qquad (7.7)$$

La ecuación lógica para la entrada J del *flip-flop* es

$$J_C = \frac{[\overline{(\overline{Load})} + \overline{(\overline{Clear})} + Q_B \cdot Q_A \cdot ENT \cdot ENP]}{\overline{\{[(\overline{(\overline{Load})} + \overline{(\overline{Clear})}) \cdot \overline{Clear} \cdot Data_C]}(\overline{(\overline{Load})} + \overline{(\overline{Clear})})\}} \qquad (7.8)$$

o bien

$$J_C = \overline{Clear}\,[\overline{Load} \cdot Q_B \cdot Q_A \cdot ENT \cdot ENP + \overline{(\overline{Load})} \cdot Data_C] \qquad (7.9)$$

Las ecuaciones lógicas 7.9 y 7.7 nos permiten determinar la tabla funcional para el contador. Cuando $\overline{Clear}$ es baja, $J_C = 0$ y $K_C = 1$, por lo que el *flip-flop* será restablecido (*reset*) en la siguiente transición positiva de un pulso en la señal de entrada *Clock*. Éste es el modo de operación *clear síncrono*. Cuando $\overline{Clear}$ es alta, los demás términos de las ecuaciones controlan el funcionamiento del *flip-flop*. Por ejemplo, si $\overline{Load}$ es baja, $J_C = Data_C$ y $K_C = \overline{Data_C}$, por lo que el valor de $Data_C$ entra en el *flip-flop* en la siguiente transición positiva de la señal *Clock*. Éste es el modo de *carga síncrona*.

El último caso por considerar es cuando $\overline{Clock}$ y $\overline{Load}$ son altas. En este caso,

$$J_C = K_C = Q_B \cdot Q_A \cdot ENT \cdot ENP \tag{7.10}$$

Cuando las señales de activación *ENT* y *ENP* son altas se está en el modo de operación *count* (contador). En este modo, el contador opera como en la figura 7.11a. El *flip-flop* Q_C alterna su valor cuando todos los bits del contador menos significativos que Q_C (en este caso, Q_B y Q_A) son altos. La salida *RCO* (acarreo de salida en cascada) se utiliza en el modo contador. La ecuación lógica para *RCO* es

$$RCO = ENT \cdot Q_D \cdot Q_C \cdot Q_B \cdot Q_A \tag{7.11}$$

En consecuencia, cuando la señal de habilitación *ENT* es alta, *RCO* es una señal de decodificación del estado en que todos los valores son iguales a uno; *RCO* es alto cuando el contador tiene el estado $(Q_D Q_C Q_B Q_A)_2 = (1111)_2$ o $(15)_{10}$. A veces, la señal *RCO* sirve para avisar que el contador está a punto de *desbordar* su intervalo dinámico (exceder su valor máximo y regresar al estado nulo).

Cuando cualquiera de las señales (ENT o ENP) es baja, $J_C = K_C = 0$, se está en el modo de operación de *retención de datos* (o *inhibición*). La figura 7.12b resume los modos de funcionamiento en una tabla funcional.

La figura 7.21c muestra un ejemplo de diagrama de tiempos. Primero se aplica un pulso *Clear* para que todos los *flip-flops* pasen al estado 0 lógico. Después se aplica un pulso *Load* para introducir un número binario en el dispositivo. En este caso, el número es $(Q_D Q_C Q_B Q_A)_2 = (1100)_2$. Observe que no se requiere el pulso *Clear* para cargar un número en este dispositivo. El pulso de carga lleva a cabo esta función sin aplicar primero el pulso *Clear*. Después se activan las señales de habilitación *ENT* y *ENP* y se aplica una serie de pulsos de conteo a la entrada *Clock*. El contador pasa de manera cíclica por sus estados (en decimal) como sigue: 12, 13, 14, 15, 0, 1 y 2. En este punto, la señal *ENP* primero y *ENT* después asumen el valor 0 e inhiben cualquier cambio posterior de estado del contador. Observe que la señal *RCO* es alta mientras el contador se encuentra en el estado en el que todos los valores son iguales a uno.

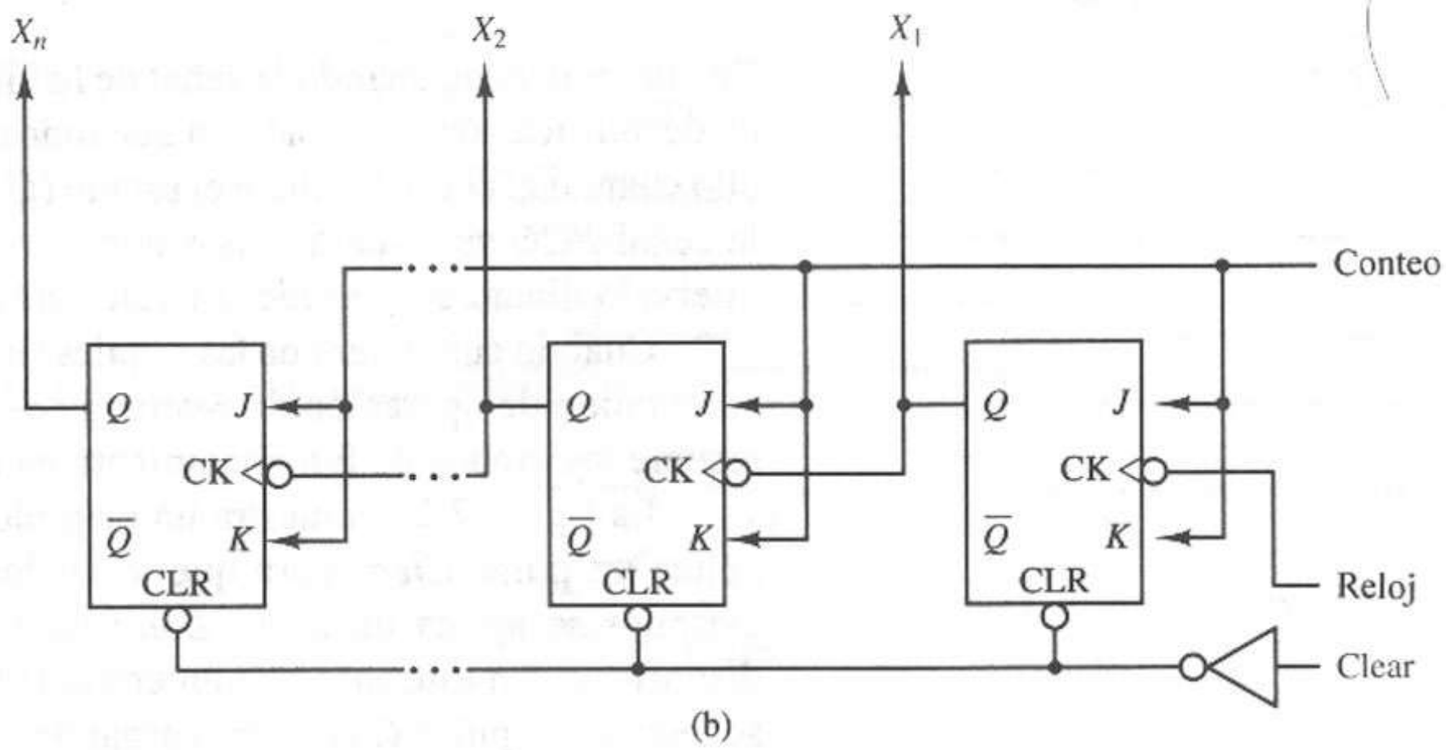
7.3.2 Contadores binarios asíncronos

Un contador binario asíncrono es aquel cuyos cambios de estado no están controlados por un pulso de reloj sincronizado. Al eliminar la necesidad de la sincronización del reloj, se puede utilizar una cantidad menor de circuitos para implantar un contador binario. Consideremos el diseño síncrono de la figura 7.11a. Podemos eliminar las compuertas AND del diseño síncrono observando las transiciones de estado del contador desde otro punto de vista; ver la figura 7.13a. La etapa del contador X_i se complementa cada vez que el estado X_{i-1} hace una transición $1 \rightarrow 0$; la etapa X_1 siempre se complementa. La figura 7.13b muestra un contador basado en estas observaciones. Podemos utilizar una orden *Clear* común asíncrona para inicializar el contador en el estado 0, y mantenemos la orden de control *Count* en 1 lógico para el conteo; el 0 lógico en *Count* inhibe todos los conteos y deja al contador en un estado constante; éste es el modo de retención de datos.

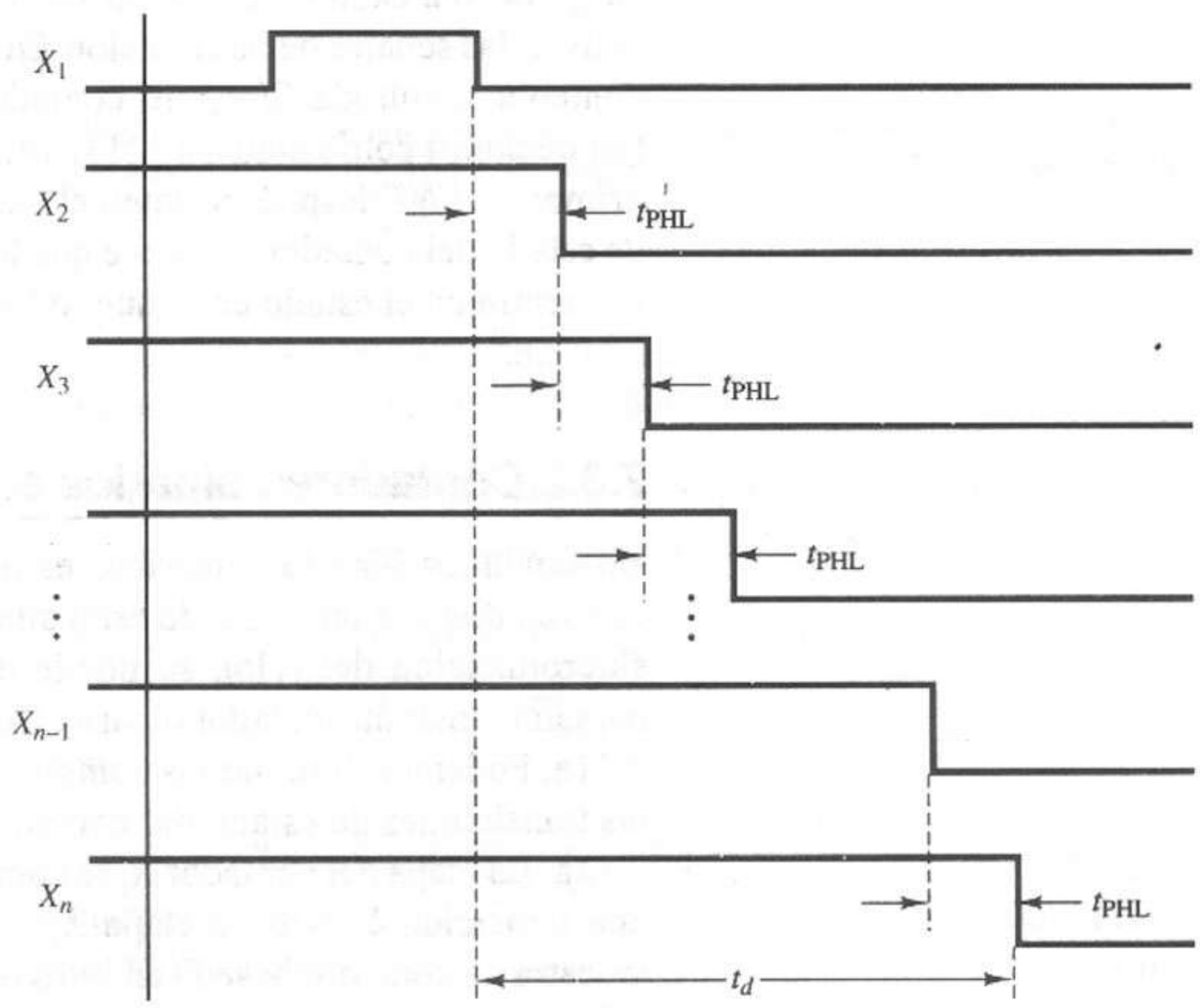
Examinemos ahora el comportamiento del contador binario asíncrono cuando sucede un desbordamiento. Justo antes del desbordamiento, todas las

X_n	...	X_3		X_2		X_1
0	...	0		0		0
0	...	0		0		1
0	...	0		1	←	0
0	...	0		1		1
0	...	1	←	0	←	0

(a)



(b)



(c)

Figura 7.13 Contador binario asíncrono. **(a)** Secuencia de estados. **(b)** Diagrama lógico. **(c)** Diagrama de tiempos.

etapas del contador asumen el valor 1 lógico, como se muestra en la figura 7.13c. Después de un pulso de reloj, el *flip-flop* de la etapa del contador X_i responde en t_{PHL} segundos. Entonces, cada etapa continúa de manera similar, hasta que todo el contador alcanza el estado lógico 0. El lector debe observar la condición transitoria producida por esta secuencia de conteo. En vez del cambio de estado deseado $(2^n-1)_{10}$ a $(0)_{10}$, el contador ha pasado por la siguiente secuencia de estados:

$$(2^n - 1)_{10}\,(2^n - 2)_{10}\,(2^n - 4)_{10}\,(2^n - 8)_{10} \cdots (2^{n-1})_{10}\,(0)_{10}$$

Aunque estas transiciones son rápidas, pueden generar condiciones transitorias no deseadas si las salidas del contador se utilizan para controlar un circuito lógico combinatorio. Repase el análisis de riesgos del capítulo 3. Por este comportamiento transitorio, los contadores asíncronos se conocen como contadores *en cascada*.

SN74293

Este módulo es una implantación en la serie SN7400 del contador binario asíncrono de la figura 7.13. La figura 7.14a muestra su diagrama lógico. Observe las dos líneas *clear* comunes $R_{0(1)}$ y $R_{0(2)}$. Cuando ambas son altas, el dispositivo pasa al estado nulo. El contador está dividido en dos segmentos, un contador de 1 bit y uno de 3 bits, y pueden estar en cascada (conectando Q_A a la entrada B) para formar un contador de 4 bits. Las entradas J y K de cada *flip-flop* están conectadas en alto dentro del dispositivo para que los *flip-flops* actúen como *flip-flops toggle* (T).

La figura 7.14b muestra el diagrama de estados del SN74293. Los círculos mayores representan los estados estables del dispositivo, los más pequeños representan un comportamiento rápido, transitorio. Observe que la transición del estado 15 al estado 0 tiene el comportamiento descrito en la figura 7.13c.

Es importante comentar ciertos aspectos acerca de este comportamiento transitorio. En primer lugar, si los pulsos del contador son mucho más lentos que el retardo del *flip-flop* JK con reloj, el contador pasará por los estados transitorios con rapidez y permanecerá la mayor parte del tiempo en los estados estables deseados. Una segunda observación es que todos los estados transitorios tienen números pares; por tanto, los estados con números nones tienen un comportamiento estable, aun en contadores de tipo cascada.

SN74177

Este módulo es una versión funcional ampliada del SN74293. Ver la figura 7.15a. En este dispositivo, la entrada A es *Clock* 1 y la entrada B es *Clock* 2. Su característica adicional es el modo de operación de *carga asíncrona*. Examine las ecuaciones lógicas para las terminales *Preset* y *Clear* de un *flip-flop* típico, digamos B:

$$Preset_B = Data_B \cdot [\overline{(Count/\overline{Load})} + \overline{(\overline{Clear})}] \cdot \overline{Clear} \qquad (7.12)$$

o bien

$$Preset_B = Data_B \cdot \overline{(Count/\overline{Load})} \cdot \overline{Clear} \qquad (7.13)$$

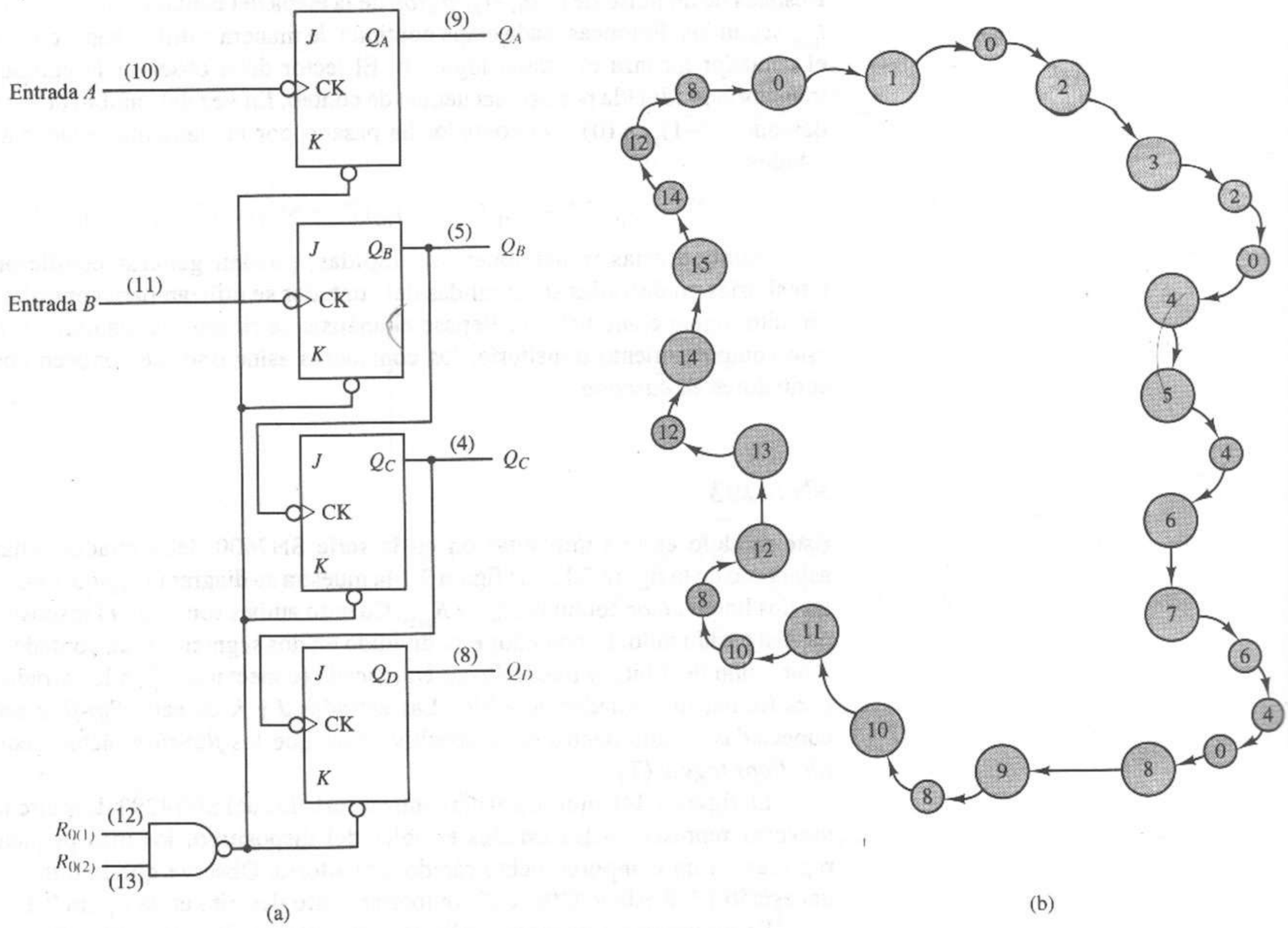


Figura 7.14 Contador binario asíncrono SN74293. **(a)** Diagrama lógico. **(b)** Diagrama de estados. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

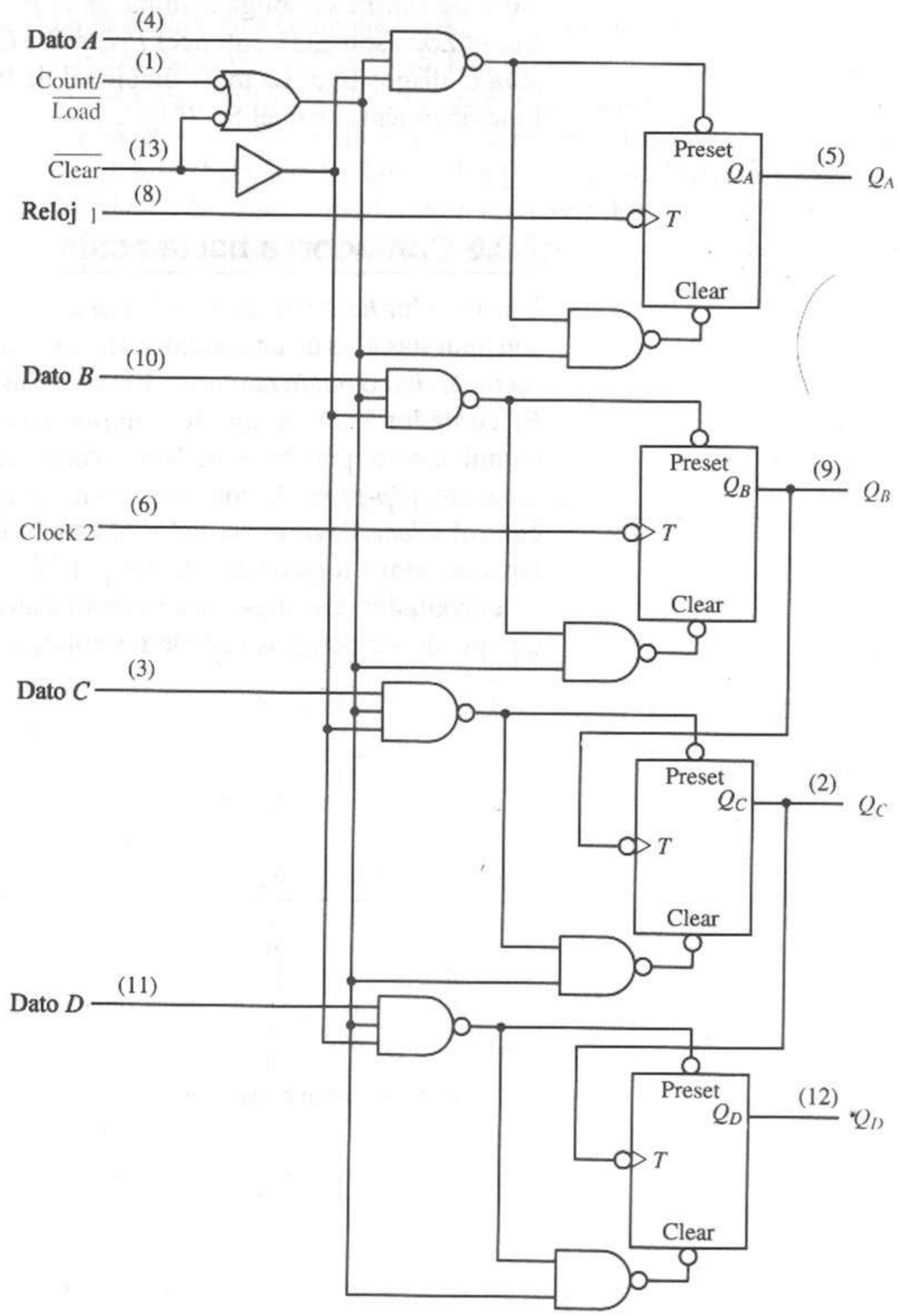
No confunda la línea de entrada $\overline{clear}$ externa y las señales *clear* para cada *flip-flop*. Las señales *clear* del *flip-flop* tienen subíndices en las siguientes ecuaciones. Para la terminal *clear* del *flip-flop B*,

$$Clear_B = \overline{[\overline{(Count/\overline{Load})} + \overline{(\overline{Clear})}] \cdot [Data_B \cdot \overline{(Count/\overline{Load})} \cdot \overline{Clear}]} \tag{7.14}$$

o bien

$$Clear_B = \overline{(\overline{Clear})} + \overline{(Count/\overline{Load})} \cdot \overline{(Data_B)} \tag{7.15}$$

Si examinamos las ecuaciones 7.13 y 7.15, vemos que cuando la línea de entrada externa $\overline{Clear}$ es baja, dentro del *flip-flop*, *PresetB* es baja y *ClearB* es alta. Éste es el modo de operación *clear común asíncrono*. Al hacer la línea externa $\overline{Clear}$ alta (inactiva), la línea externa $Count/\overline{Load}$ puede controlar el dispositivo.



(a)

Entradas		
$\overline{\text{Clear}}$	Count/$\overline{\text{Load}}$	Modo
L	×	*Clear* común asíncrono
H	L	Carga asíncrona
H	H	Conteo

(b)

Figura 7.15 Contador binario asíncrono SN74177. **(a)** Diagrama lógico. **(b)** Tabla funcional. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

Si la línea *Clear* es alta y *Count*/$\overline{Load}$ es baja, entonces *PresetB* = *DataB* y *ClearB* = ($\overline{DataB}$). Éste es el modo de operación de *carga asíncrona*, ya que el valor de *DataB* se obliga a entrar en el *flip-flop*. Si las líneas externas $\overline{Clear}$ y *Count*/$\overline{Load}$ son altas, entonces *PresetB* = *ClearB* = 0; éste es el modo de *conteo* para el dispositivo. La tabla funcional de la figura 7.15b resume los modos de funcionamiento para el SN74177.

7.3.3 Contadores hacia abajo

Un contador *hacia abajo* o en *hacia atrás* es aquel cuyas transiciones de estado son opuestas a las de un contador estándar, que también se conoce como contador *hacia arriba* o *hacia adelante*. Examine las tablas de estados de la figura 7.16a. El contador hacia abajo se comporta como un contador hacia arriba con complemento; por tanto, podemos construir un contador hacia abajo asíncrono mediante *flip-flops* JK con reloj, como se indica en la figura 7.16b. La señal de control *Clear* lleva el contador al estado 0, y la señal de control *Count* debe tener el valor 1 lógico para que los pulsos del reloj provoquen cambios de estado en el contador. De nuevo, este diseño asíncrono produce un efecto de cascada, que puede ser peligroso en ciertas aplicaciones.

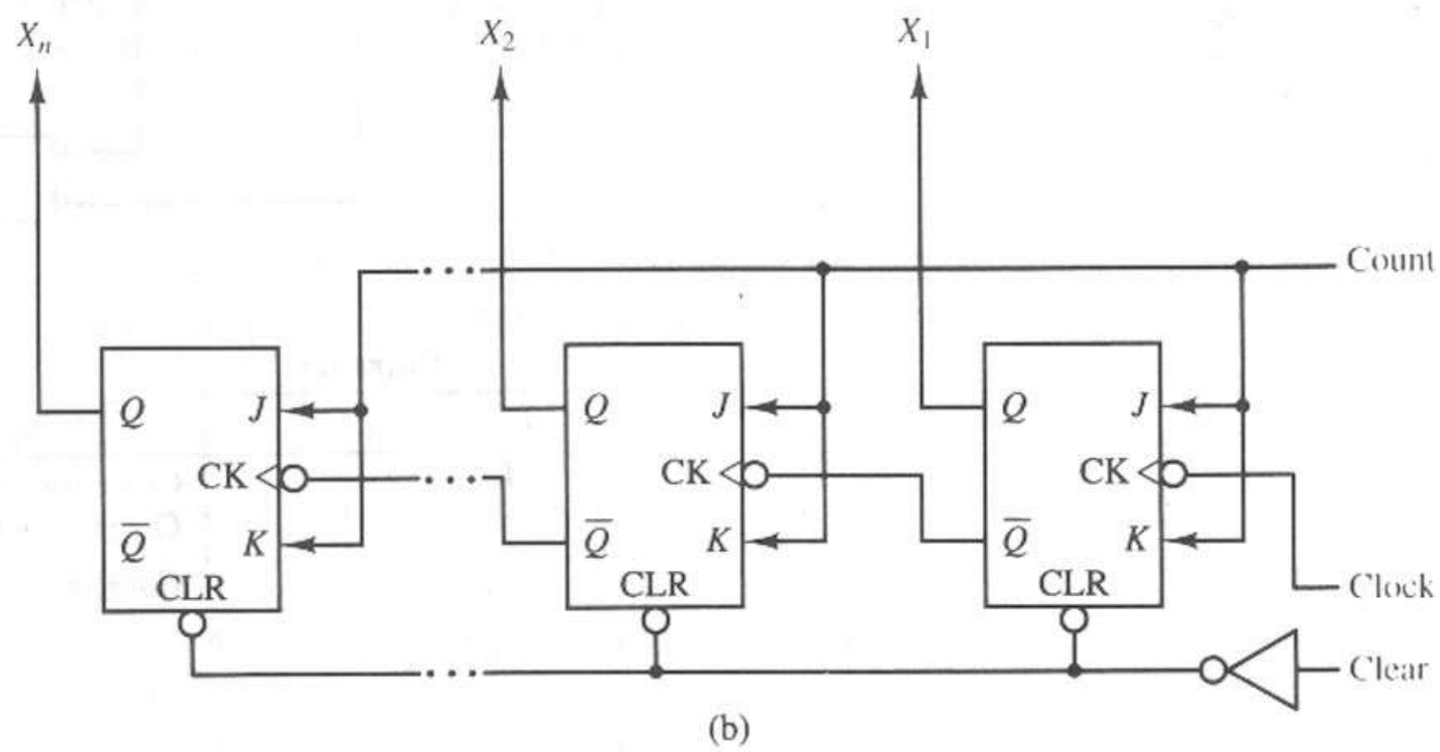
X_n	...	X_3	X_2	X_1
1	...	1	1	1
0	...	0	0	0
0	...	0	0	1
0	...	0	1	0
0	...	0	1	1
0	...	1	0	0

Modo de conteo hacia arriba

X_n	...	X_3	X_2	X_1
0	...	0	0	0
1	...	1	1	1
1	...	1	1	0
1	...	1	0	1
1	...	1	0	0
1	...	0	1	1

Modo de conteo hacia abajo

(a)



(b)

Figura 7.16 Contador hacia abajo asíncrono. **(a)** Secuencia de estados. **(b)** Diagrama lógico.

7.3.4 Contadores hacia arriba/hacia abajo

Muchos sistemas digitales requieren un diseño de contador que pueda funcionar tanto hacia arriba como hacia abajo. La figura 7.17 muestra una combinación de contador síncrono arriba/abajo. Este contador puede estar en ambos modos, pues la señal de control hacia abajo es el complemento de la señal de control hacia arriba. Por tanto, rotulamos la línea de control arriba/abajo como *Arriba/$\overline{\text{abajo}}$*. En el modo hacia arriba, las salidas Q de los *flip-flops* controlan las terminales J y K de los *flip-flops* superiores en la cadena. En el modo hacia abajo, las salidas $\overline{Q}$ de los *flip-flops* desempeñan este papel.

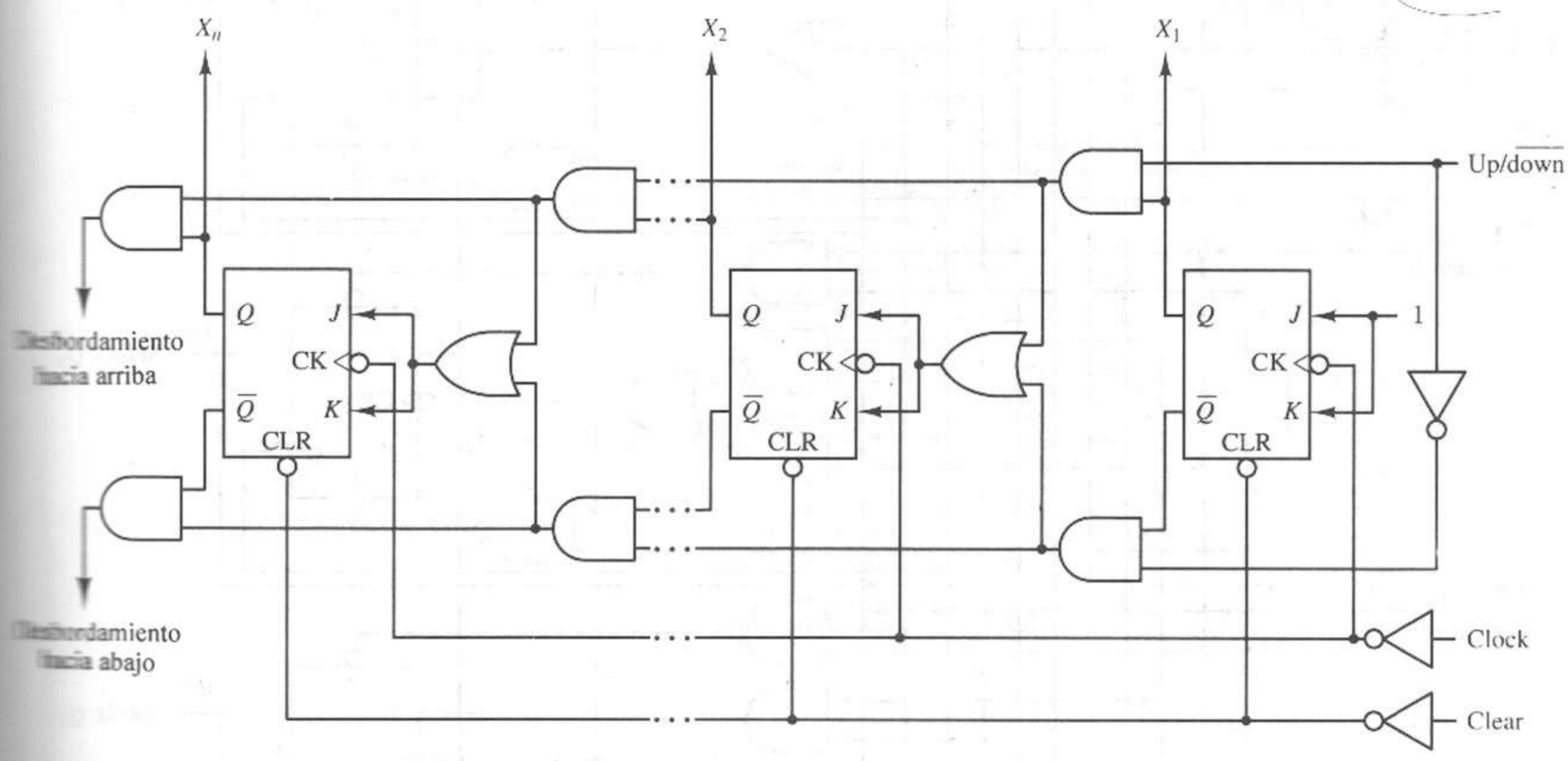


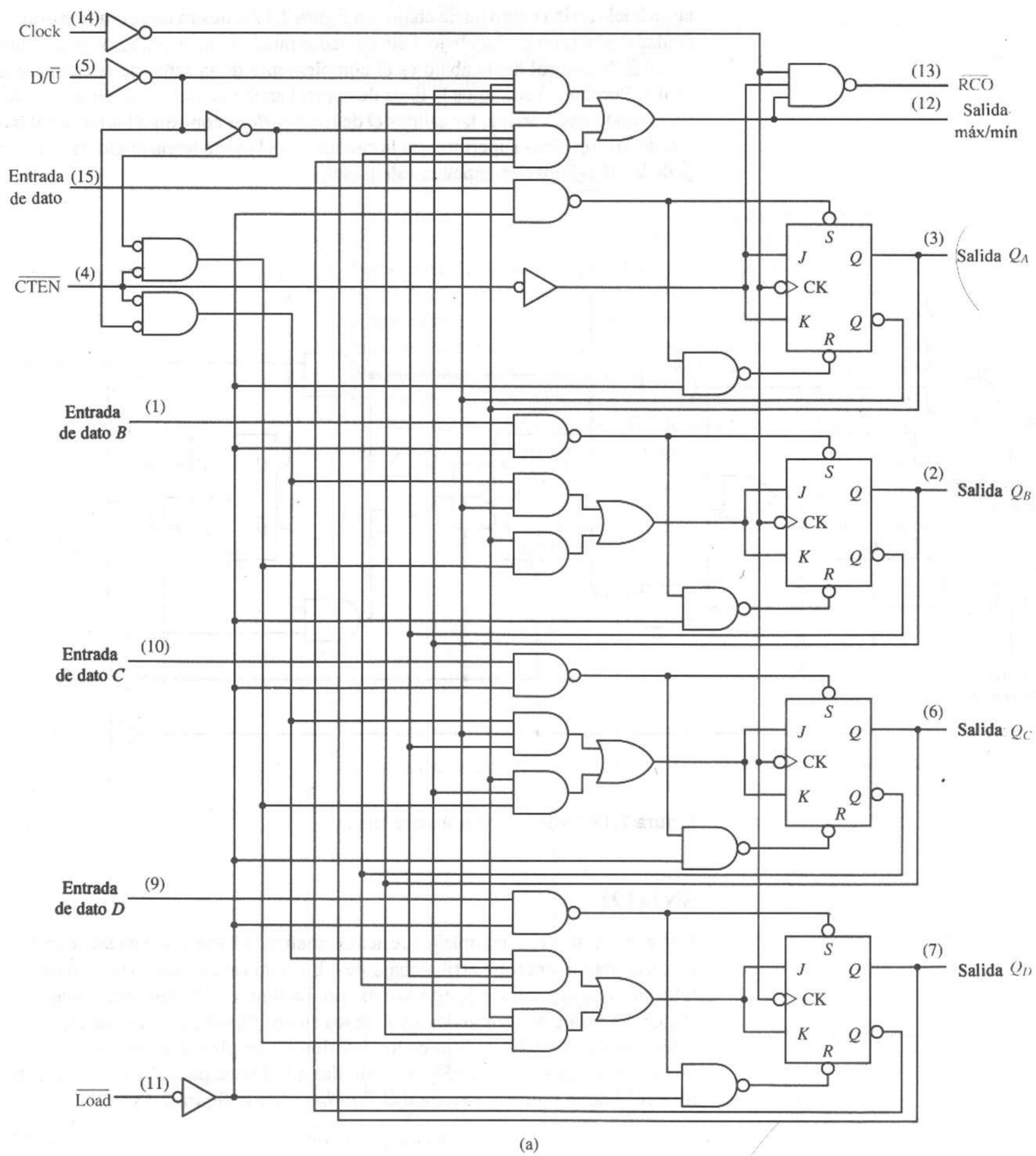
Figura 7.17 Contador arriba/abajo síncrono.

SN74191

Éste es el circuito más complejo que hemos analizado hasta ahora en este capítulo. Es un contador síncrono arriba/abajo de 4 bits con carga asíncrona, activación, reloj en cascada y salidas de estado máximo. La figura 7.18a muestra el diagrama lógico. Examine las terminales S y R de los cuatro *flip-flops*; éstas corresponden a las terminales *Preset* y *Clear* en los anteriores ejemplos de circuitos. La lógica que controla las terminales S y R es similar a la lógica para *Preset* y *Clear* del SN74177 de la figura 7.15a. Para el *flip-flop C* de la figura 7.18a,

$$S_C = Data_C \cdot \overline{(\overline{Load})} \tag{7.16}$$

$$\begin{aligned} R_C &= \overline{[Data_C \cdot \overline{(\overline{Load})}] \cdot \overline{(\overline{Load})}} \\ &= \overline{Data_C} \cdot \overline{(\overline{Load})} \end{aligned} \tag{7.17}$$

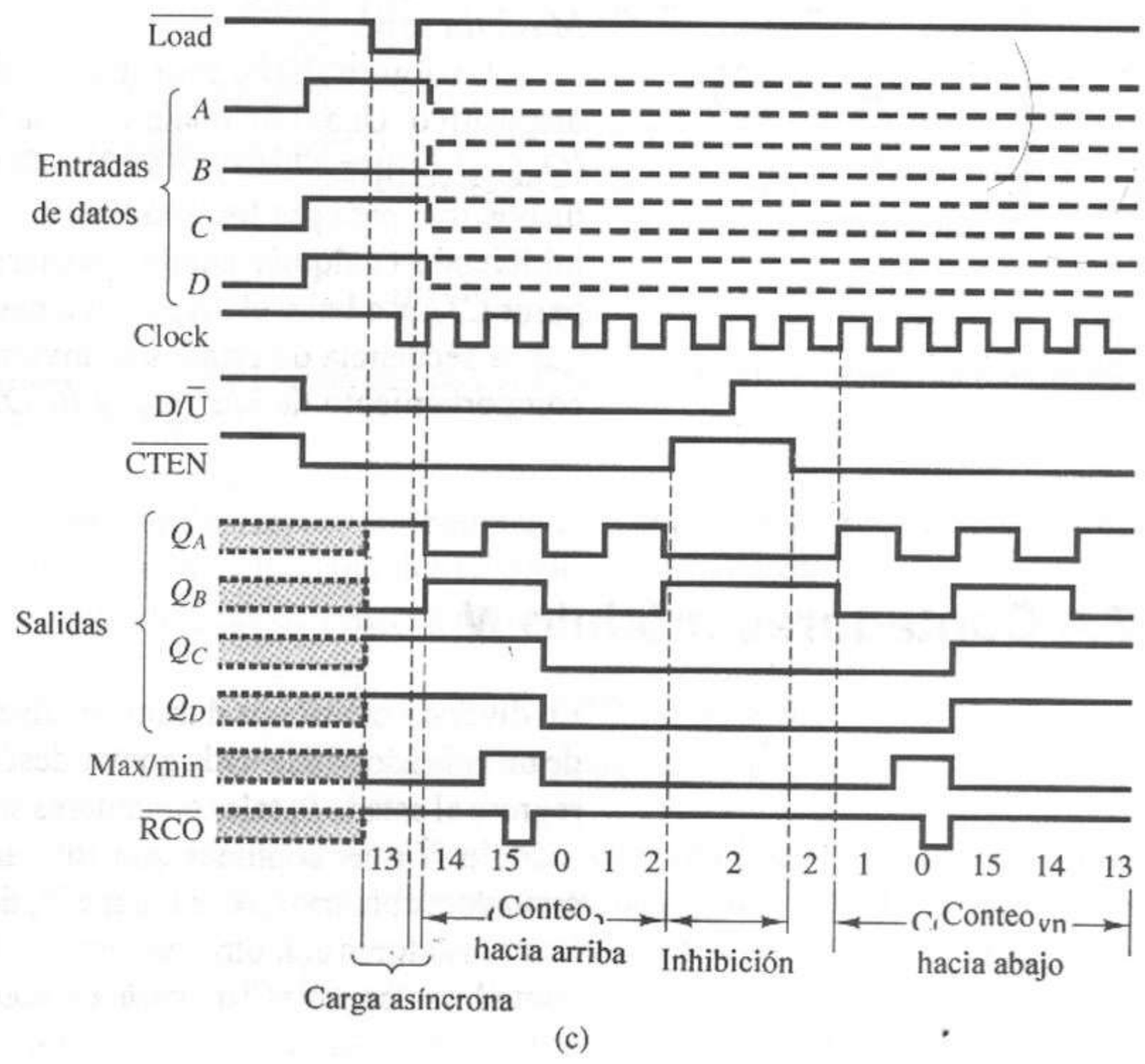


(a)

Figura 7.18 Contador arriba/abajo SN74191. **(a)** Diagrama lógico.

Entradas			
$\overline{Load}$	$\overline{CTEN}$	$D/\overline{U}$	Modo
L	×	×	Carga asíncrona
H	H	×	Retención de datos (inhibición)
H	L	L	Conteo hacia arriba
H	L	H	Conteo hacia abajo

(b)



(c)

Figura 7.18 (continuación) Contador arriba/abajo SN74191. **(b)** Tabla funcional. **(c)** Diagrama de tiempos. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

Cuando $\overline{Load}$ es baja, tenemos que $S_C = Data_C$ y $R_C = (\overline{Data_C})$, de modo que el valor de $Data_C$ se carga en el *flip-flop* de manera asíncrona. Cuando $\overline{Load}$ es alta, $S_C = R_C = 0$, de modo que el contador es controlado por sus otras entradas, $\overline{CTEN}$, $D/\overline{U}$ y *Clock*.

Examinemos ahora las entradas J y K del *flip-flop* C:

$$J_C = K_C = \overline{(\overline{CTEN})} \cdot [Q_B \cdot Q_A \cdot \overline{(D/\bar{U})} + \bar{Q}_B \cdot \bar{Q}_A \cdot (D/\bar{U})] \qquad (7.18)$$

Cuando la señal de entrada de activación de conteo $\overline{CTEN}$ es alta, J_C y K_C tienen el valor 0 lógico, por lo que no habrá cambio alguno en las salidas del *flip-flop*. Cuando $\overline{CTEN}$ es baja, el contador entra en el modo de conteo hacia arriba o hacia abajo, según el valor de $D/\overline{U}$. Si $D/\overline{U}$ es baja, entonces Q_B y Q_A determinan el valor alternante de Q_C de modo que el contador esté en el modo *hacia arriba*.

Cuando $D/\overline{U}$ es alta, $\overline{Q_B}$ y $\overline{Q_A}$ determinan el valor alternante de QC, por lo que el contador pasa por su secuencia de estados hacia atrás, o *hacia abajo*. La figura 7.18b muestra la tabla funcional de este dispositivo.

Una característica adicional de este contador es su señal de salida *Max/min*. En el modo hacia arriba, esta señal es alta en el estado 15. En el modo hacia abajo, es alta en el estado 0. La señal de salida *Max/min* también alimenta la salida $\overline{RCO}$:

$$\overline{RCO} = \overline{[\overline{Clock} \cdot (Max/min) \cdot \overline{(\overline{CTEN})}]} \tag{7.19}$$

$\overline{RCO}$ es baja cuando el dispositivo está contando ($\overline{CTEN}$ es baja) y la salida *Max/min* sube.

La figura 7.18c muestra un ejemplo de diagrama de tiempos para el dispositivo. Una funcionamiento de carga asíncrona lleva el contador al estado $(Q_DQ_CQ_BQ_A)_2 = (1101)_2 = (13)_2$. Entonces, como $D/\overline{U}$ y $\overline{CTEN}$ son bajas, el dispositivo pasa por los estados 14, 15, 0, 1 y 2. Después, la señal $\overline{CTEN}$ es alta, inhibiendo cualquier cambio posterior hacia arriba. Al cambiar $D/\overline{U}$ a alta y pasar $\overline{CTEN}$ a baja, el dispositivo pasa al modo de conteo hacia abajo, de modo que la secuencia de estados se invierte: 2, 1, 0, 15, 14, 13, etcétera. Observe el comportamiento de *Max/min* y $\overline{RCO}$ en el diagrama de tiempos.

7.4 Contadores módulo *N*

En diversas ocasiones durante el diseño de sistemas digitales surge la necesidad de un contador que pueda contar desde el estado 0 hasta el estado $N-1$ y después regrese al estado 0; tales contadores son los contadores *módulo N*. Los contadores módulo N más comunes son los binarios, que ya hemos analizado. Para los contadores binarios, N es igual a 2^n, donde n es el número de etapas del contador. Los contadores con otros valores de N también son muy útiles. Por ejemplo, los contadores con $N = 10$ aparecen con frecuencia en el diseño de sistemas digitales.

7.4.1 Contadores BCD síncronos

Un contador BCD (decimal codificado en binario) síncrono es un contador módulo 10, o de *décadas*. El contador BCD debe comportarse como un contador binario hasta llegar a la etapa 9. En este momento, los circuitos de control deben preparar las entradas del *flip-flop* de modo que el siguiente pulso de reloj obligue al contador a regresar al estado (0000) en vez de permitir que llegue al siguiente estado del contador binario (1010). El módulo SN74160 tiene un diseño de contador BCD asíncrono.

SN74160

Éste es un contador síncrono de décadas con carga síncrona, *clear*, activación y acarreo de salida en cascada asíncronos. Ver figura 7.19.

La señal de control del *clear* asíncrono $\overline{Clear}$ es activa baja. La señal de control de carga $\overline{Load}$ también es activa baja. ¿Cómo opera este contador? Podemos analizar sus características operativas examinando un *flip-flop* típico

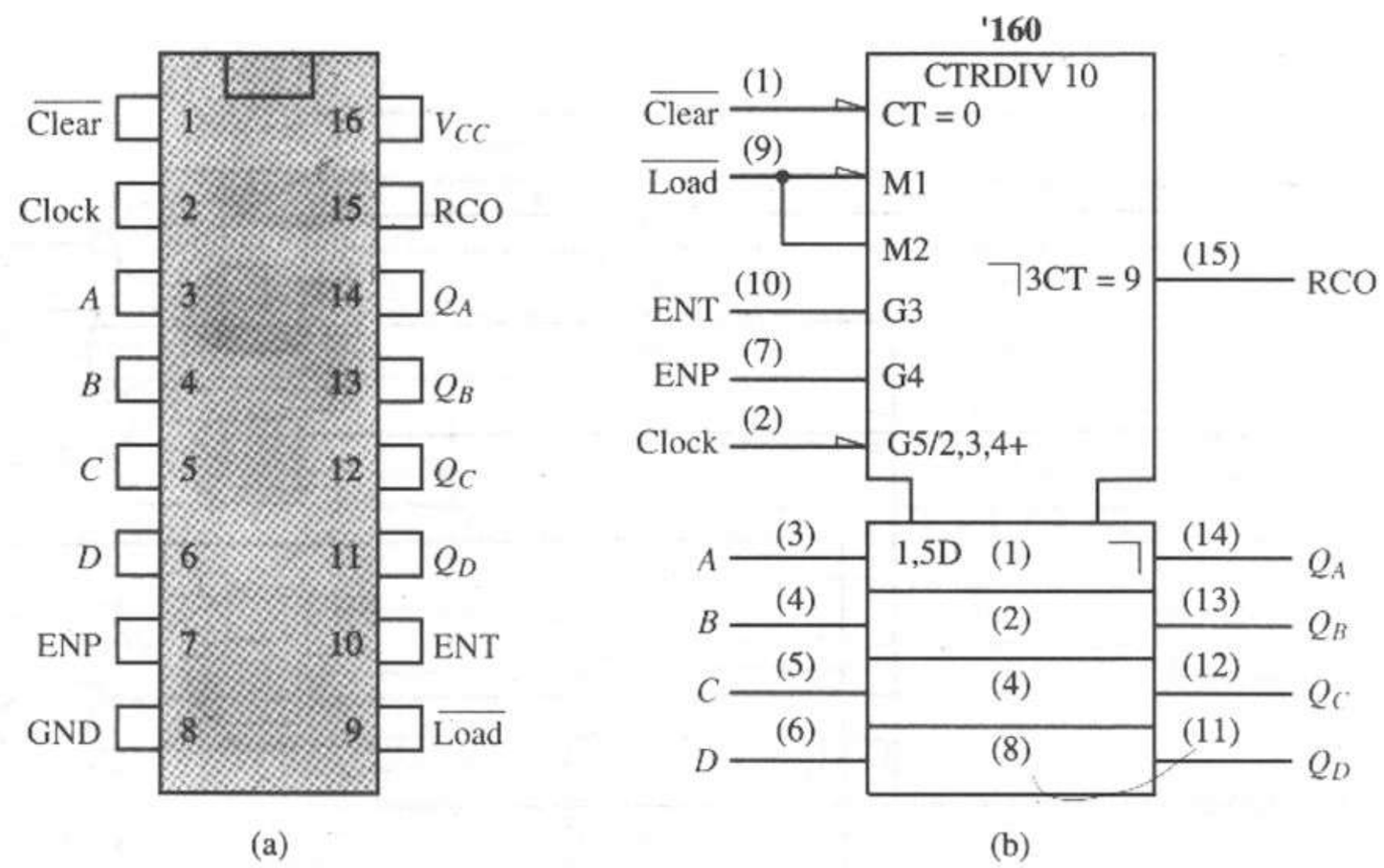


Figura 7.19 Contador de décadas síncrono SN74160. **(a)** *Pines* del paquete. **(b)** Símbolo lógico.

y generalizando después los resultados al contador de 4 bits. Supongamos que decidimos contar la etapa del contador Q_B y analizar sus entradas J y K. Las ecuaciones lógicas de estas entradas son

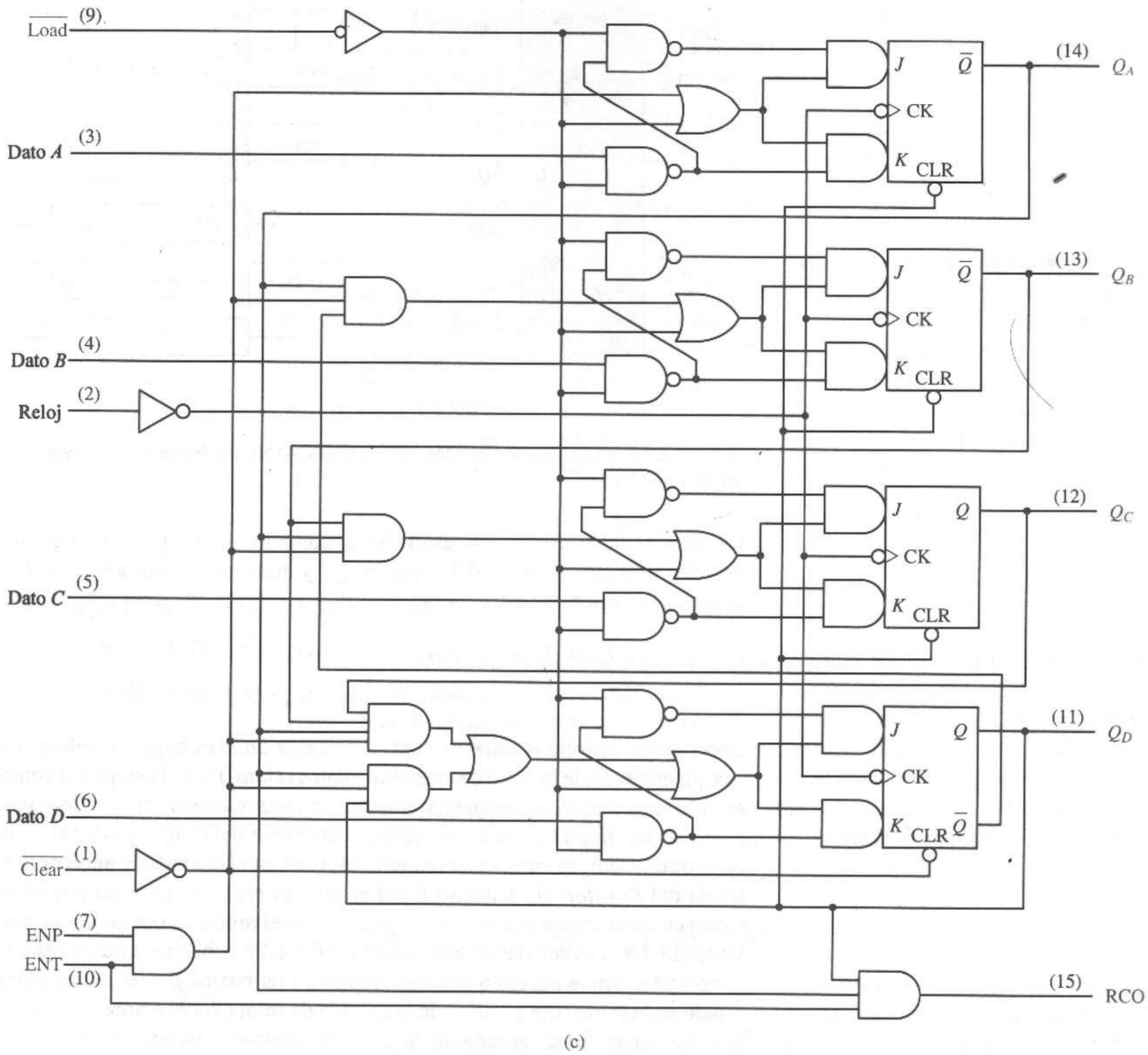
$$J_B = \overline{(\overline{Load})} \cdot Data_B + \overline{Load} \cdot Q_A \cdot \bar{Q}_D \cdot ENT \cdot ENP$$
$$K_B = \overline{(\overline{Load})} \cdot \overline{Data_B} + \overline{Load} \cdot Q_A \cdot \bar{Q}_D \cdot ENT \cdot ENP$$

Observe que cuando la señal de control de carga $\overline{Load}$ es baja, los valores real y complementado de la señal de entrada de datos ($Data_B$) para la etapa del contador se colocan en J_B y K_B, respectivamente. En consecuencia, el *flip-flop* actuará como un *flip-flop* D con reloj, y, al siguiente pulso del reloj, el valor en la línea de entrada $Data_B$ se cargará de manera síncrona en el contador y aparecerá en la salida del *flip-flop* Q_B. Cuando $\overline{Load}$ es alta, el segundo término producto de cada ecuación lógica controla a J_B y K_B. Éste es el modo de operación de conteo síncrono. Las dos señales de activación *ENP* y *ENT* deben ser altas antes de que el contador inicie un ciclo con sus estados. Los cuatro *flip-flops* cambian de estado en la transición positiva de la entrada de reloj *Clock*. Ahora, examinemos la ecuación lógica de la señal de acarreo de salida en cascada *RCO*

$$RCO = Q_D \cdot Q_A \cdot ENT$$

Puesto que el contador forma un ciclo con los estados 0, 1, 2,...,9, los estados 10, 11,..., 15 nunca ocurren. Por tanto, el estado máximo del contador (1001) se puede detectar mediante una compuerta AND con entradas Q_D y Q_A. La señal de entrada *ENT* debe estar alta para activar *RCO*.

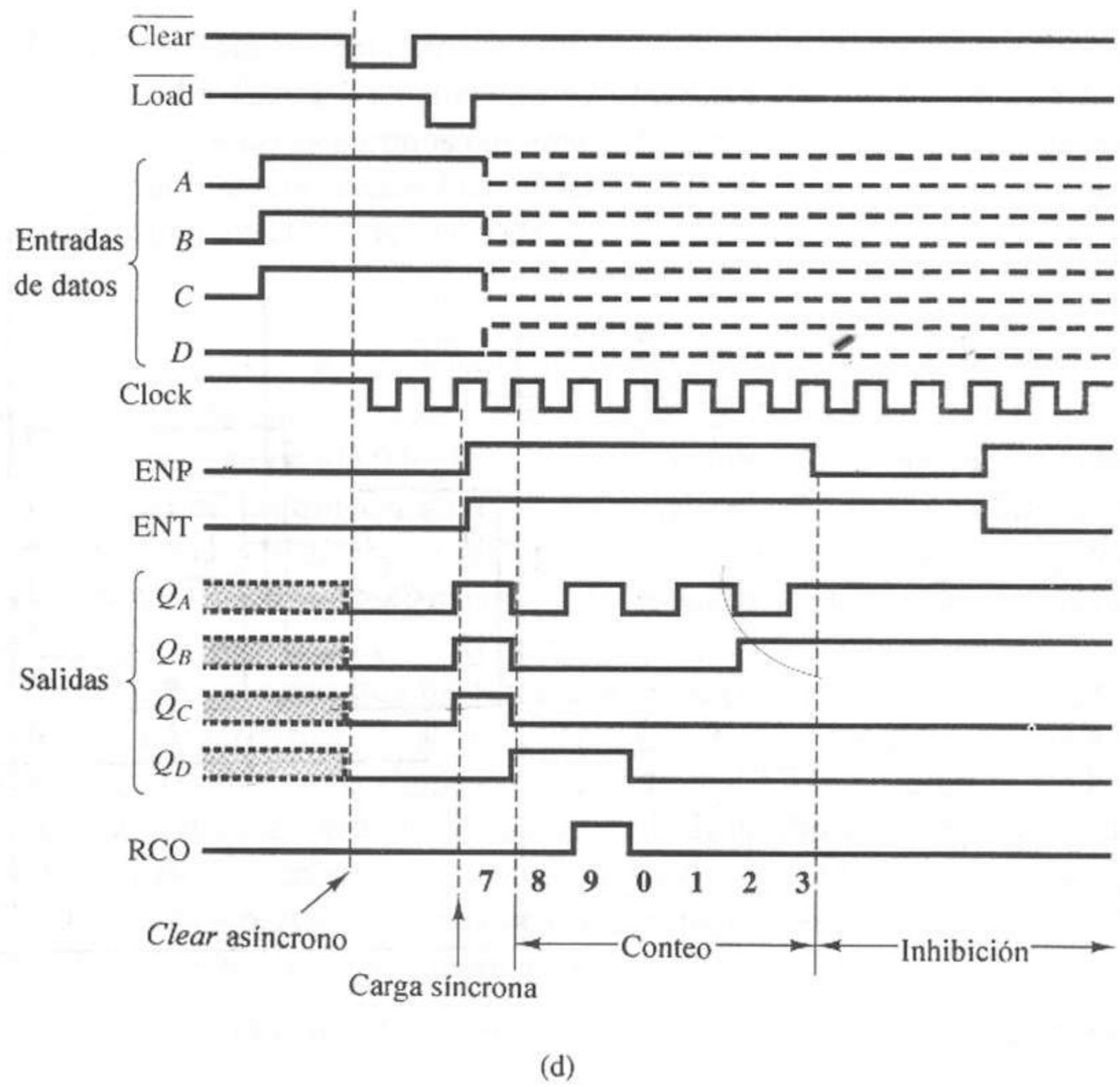
La figura 7.19d muestra un diagrama de tiempos para una secuencia típica de operaciones del SN74160. Primero aplicamos una señal *clear* asíncrona para llevar el contador al estado nulo. A continuación, una funcionamiento de carga síncrona lleva el contador al estado 7 (0111). Después aplicamos las dos señales de activación, *ENT* y *ENP*, para iniciar el modo de operación de conteo síncrono. En la transición positiva de cada pulso de reloj *Clock*, el contador pasa al estado



(c)

Figura 7.19 (continuación) Contador de décadas SN74160. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

8, luego al 9, y después regresa al estado 0, y continúa con los estados 1, 2 y 3. En este momento, la señal de habilitación *ENP* se hace baja e inhibe la secuenciación posterior del contador. Observe que podemos utilizar cualquier señal *ENP* o *ENT* para inhibir el contador (colocarlo en el modo de retención de datos). Observe también que la señal *RCO* es alta durante el periodo en que el contador está en el estado 9, su estado máximo.



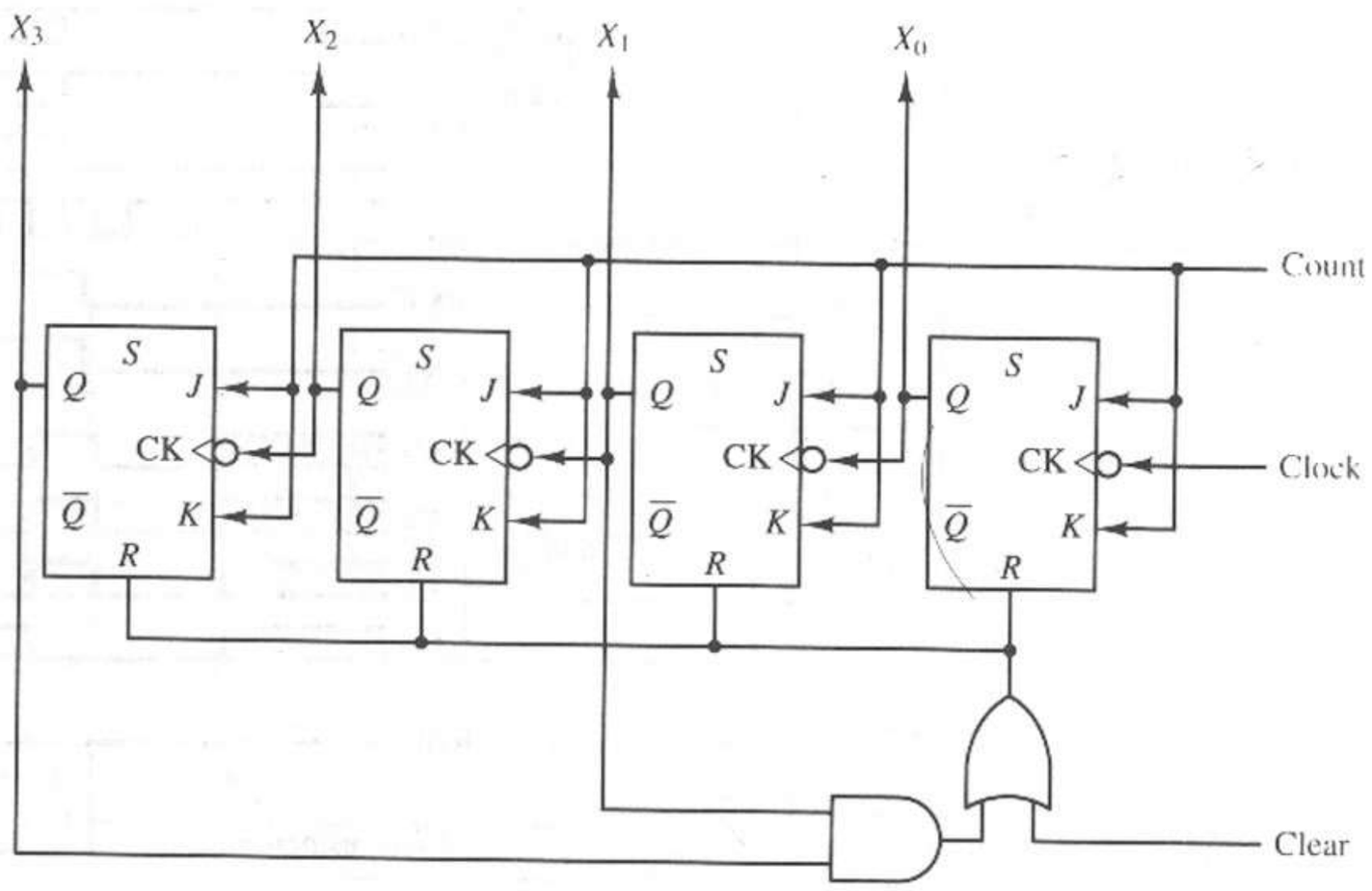
(d)

Figura 7.19 (continuación) Contador de décadas SN74160. **(d)** Diagrama de tiempos. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

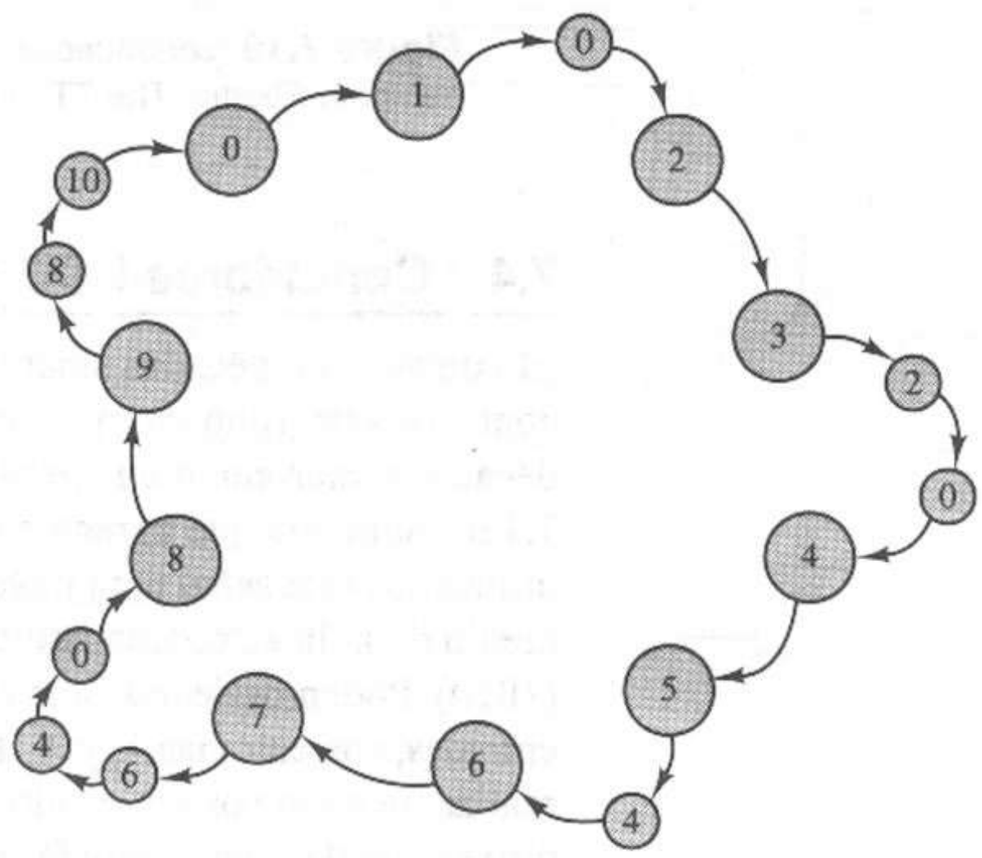
7.4.2 Contadores BCD asíncronos

El contador de décadas síncrono de la figura 7.19 puede reemplazarse por un contador asíncrono en muchas aplicaciones. Podemos diseñar un contador de décadas asíncrono, o *en cascada*, modificando el contador binario de la figura 7.13b. Suponga que agregamos un circuito lógico para detectar el estado 10 y utilizamos esta señal para llevar el contador en forma inmediata al estado 0 por medio de la línea común *reset*. El estado 10 tiene el valor binario $(X_3X_2X_1X_0) = (1010)$. Podemos decodificar el estado 10 mediante una compuerta AND de dos entradas, con entradas X_3 y X_1. La condición $X_3 = X_1 = 1$ es única, pues el contador realiza un ciclo por los estados 0, 1, 2,..., 9 antes de llegar al estado 10. Ningún otro estado de la serie satisface la condición $X_3 = X_1 = 1$. La figura 7.20a muestra el diagrama lógico para el contador de décadas asíncrono. Ahora analizaremos el comportamiento transitorio de este diseño de contador en cascada.

En la figura 7.20b utilizamos un diagrama de estados para describir el comportamiento de este circuito. Ya describimos antes en este capítulo el comportamiento transitorio del contador de los estados 0 a 9. (Ver Fig. 7.14b.) En esta figura, los círculos pequeños representan los estados transitorios. La condición transitoria en el peor de los casos ocurre en la transición del estado 7 al 8. Debido al efecto de cascada, se observan tres estados intermedios. Ahora, consideremos la transición del estado 9 al estado 0. El efecto de cascada hace que el contador



(a)



(b)

Figura 7.20 Contador BCD asíncrono. **(a)** Diagrama lógico. **(b)** Diagrama de estados.

entre en el estado 8 y después en el estado 10. Sin embargo, el circuito de retroalimentación detectará el estado 10 y generará una señal *clear* común para regresar todas las etapas del contador al valor 0 lógico.

SN74176

SN74176 es un módulo contador asíncrono que puede realizar las operaciones ÷2, ÷5 o ÷10. La figura 7.21 muestra su diagrama lógico. El módulo tiene controles *clear* y *load* asíncronos comunes. Podemos explicar las operaciones *load* y *clear* del contador si examinamos las ecuaciones lógicas para las entradas *set* y *reset* de una etapa típica, digamos Q_C:

$$S_C = \overline{Clear} \cdot \overline{(\overline{Load})} \cdot Data_C$$

$$R_C = \overline{(\overline{Clear})} + \overline{(\overline{Load})} \cdot \overline{Data}_C$$

Cuando la señal de control *clear* $\overline{Clear}$ es baja, $(S_C,R_C) = (0,1)$ y la etapa del contador regresa a su estado 0 lógico. Cuando $\overline{Clear}$ es alta (inactiva), una señal baja en la línea de control $\overline{Load}$ crea la condición $(S_C,R_C) = Data_C,\overline{Data}_C$, que mete el valor binario de $Data_C$ en la etapa del contador, lo que carga al *flip-flop*. Cuando $\overline{Clear}$ y $\overline{Load}$ son altas (inactivas), el dispositivo actúa como un contador asíncrono.

En el modo de operación de conteo, la etapa Q_A actúa como una etapa de contador ÷2. Los otros tres *flip-flops* (Q_D, Q_C, Q_B) forman una etapa de contador ÷5, donde Q_D es el bit más significativo. La figura 7.21b muestra el símbolo lógico para el contador. Observemos las entradas de los *flip-flops* Q_D y Q_B. ¡Estos dos *flip-flops* JK operan en forma síncrona! El *flip-flop* Q_C alterna cuando Q_B cambia de alto a bajo. Así, podemos describir el funcionamiento correcto de las tres etapas del contador de la manera siguiente:

1. Se ponen todos los *flip-flops* en cero con una señal baja en la entrada $\overline{Clear}$.
2. Se inicia una serie de pulsos de reloj sobre la entrada *Clock2*.
3. $\overline{Q}_D$ se mantiene alta mediante el pulso *clear*, de modo que las entradas *J* y *K* del *flip-flop* Q_B siguen siendo altas, lo que hace que Q_B actúe como un *flip-flop* con alternancia.
4. En los primeros pulsos de reloj, Q_D permanece alta mientras el par de *flip-flops* (Q_C, Q_B) actúa como un contador binario, siguiendo la secuencia de estados 0, 1, 2 y 3. En el siguiente pulso del reloj, los dos *flip-flops* Q_C y Q_B son altos, de modo que la entrada *J* de Q_D también es alta debido a la acción de la compuerta AND. Ahora analizaremos la entrada *K* de Q_D en este punto de la secuencia de estados. Como la entrada *K* de Q_D está ligada a Q_D (la salida del *flip-flop*), tiene un valor bajo, de modo que $(J_D, K_D) = (1, 0)$, la condición *set*. Cuando ocurre el siguiente pulso de reloj, Q_D se mantiene alta, mientras que Q_B y Q_C cambian su valor a bajo en la secuencia normal de conteo binario. En este momento de la serie, el contador de 3 bits comienza en el estado 0, después cambia a los estados 1, 2 y 3 y ahora se detiene en el estado 4.
5. Como Q_D es alta, $\overline{Q}_D$ es baja, de modo que las entradas *J* y *K* de Q_B son bajas e inhiben cualquier cambio en el siguiente pulso de reloj sobre *Clock2*. Puesto que Q_C y Q_B son bajas, la entrada *J* de Q_D es baja, mientras que la entrada *K* ligada a Q_D es alta, $(J_D, K_D) = (0,1)$. Ésta es la condición *reset*, de modo que Q_D cambiará a 0 en el siguiente pulso de reloj. Por tanto, el siguiente estado después del estado 4 para el contador de 3 bits será el estado nulo.
6. La serie de estados para el contador de 3 bits es $(Q_D, Q_C, Q_B) = 0, 1, 2, 3, 4, 0, 1,...$

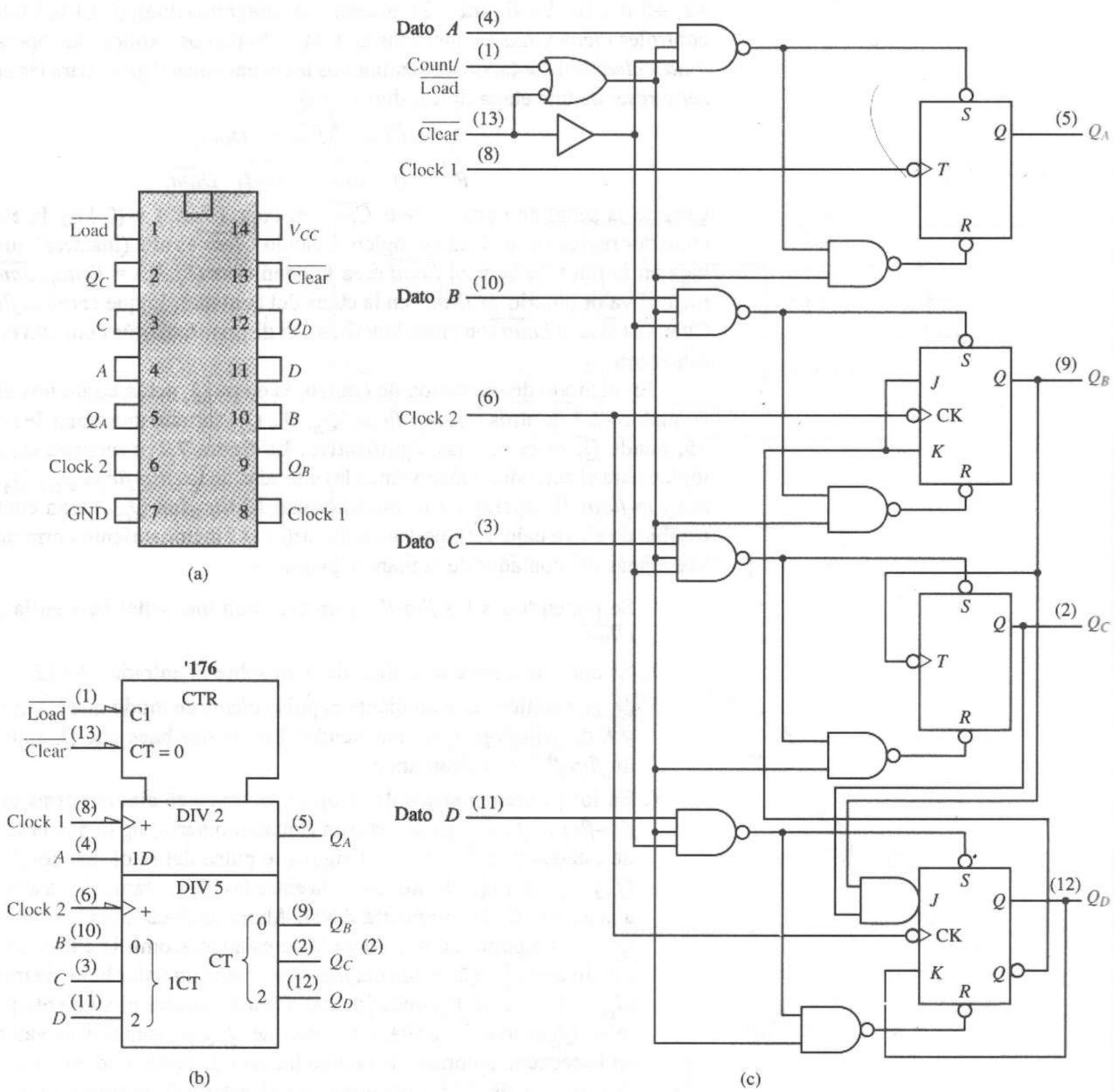


Figura 7.21 Contador de décadas asíncrono SN74176. **(a)** *Pines* del paquete. **(b)** Símbolo lógico. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

Suponga que conectamos la salida Q_A a la entrada *Clock2* y proporcionamos una serie de pulsos de conteo a la entrada *Clock1*. El *flip-flop* Q_D será el bit más significativo, mientras que el *flip-flop* Q_A será el menos significativo. Generamos la siguiente secuencia de conteo: $(Q_D, Q_C, Q_B, Q_A) = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0,$

1,... En este modo de operación, decimos que el módulo es un contador de décadas. Pero si en vez de esto conectamos la salida Q_D a la entrada *Clock1* y proporcionamos una serie de pulsos de conteo a la entrada *Clock2*, Q_A será el bit más significativo y generaremos la siguiente secuencia de conteo: (Q_A, Q_D, Q_C, Q_B) = 0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 0, 1,... Esta secuencia de conteo recibe el nombre de modo de operación *biquinario*. El contador tiene 10 estados, pero la secuencia ya no está formada por los dígitos decimales.

7.4.3 Contadores módulo 6 y módulo 12

Otros dos contadores módulo *N* tienen aplicaciones frecuentes en el diseño digital: los contadores módulo 6 y módulo 12. ¿Alguna vez se ha preguntado cómo opera un cronómetro digital? Un oscilador de alta frecuencia (o la línea de corriente de 60 hertz) proporciona una señal de reloj periódica que se introduce en una serie de contadores. Podemos utilizar un contador *módulo 10* para generar decenas de segundos (o minutos) a partir de un pulso de conteo de 1 segundo (o un minuto). Podemos utilizar un contador *módulo 6* para generar una secuencia de pulsos de un minuto a partir de la señal "decenas de segundos". La figura 7.22 muestra el diagrama de bloques de un cronómetro digital simple que utiliza la línea de corriente eléctrica para generar señales de reloj. Usamos la señal de control *Clear* para inicializar el cronómetro. Entonces podemos utilizar la señal *Start/Stop* para aplicar o inhibir los pulsos de conteo que provienen del generador de pulsos conectado a la línea de corriente. Los pulsos del generador de pulsos

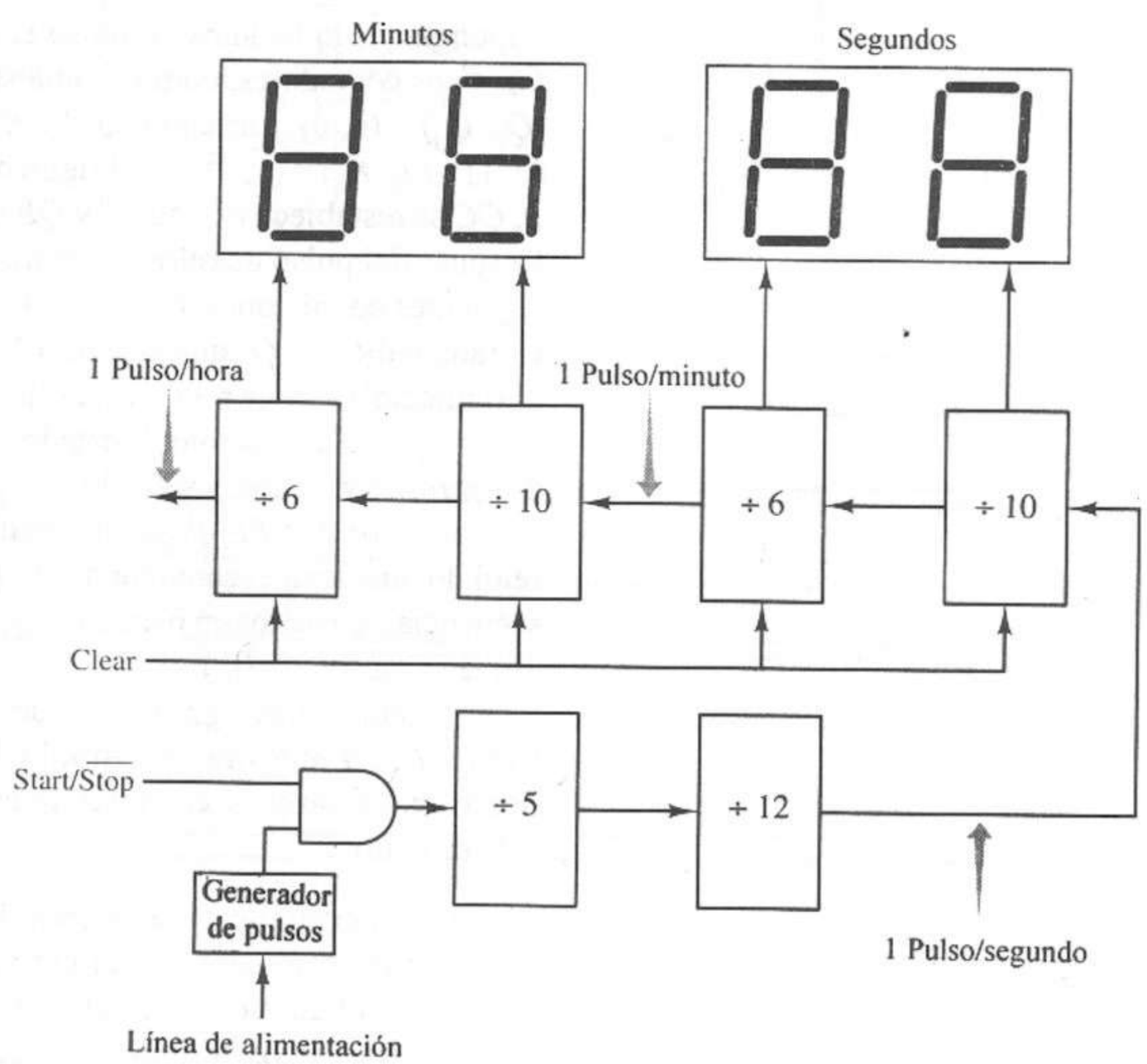


Figura 7.22 Diagrama de bloques para un cronómetro digital.

se producen a razón de 60 por segundo. Así, utilizamos un contador módulo 60 para reducir la frecuencia de los pulsos a 1 por segundo. Implantamos el contador módulo 60 mediante un contador módulo 5 en cascada con un contador módulo 12. Podríamos utilizar con la misma efectividad una combinación ÷6 y ÷10. Utilizamos otros dos contadores módulo 60 para exhibir segundos y minutos. Si se necesitan horas en el diseño, basta con agregar otra etapa ÷10 y ÷6. Podemos implantar los contadores módulo 5 y módulo 10 mediante el SN74176, y los contadores módulo 6 y módulo 12 mediante el contador SN7492A que describiremos a continuación.

SN7492A

El SN7492A es un contador síncrono módulo 6 o módulo 12 con un *clear* asíncrono común. La figura 7.23 describe este dispositivo. Podemos determinar su funcionamiento a partir del diagrama lógico de la figura 7.23c.

Primero analizaremos la etapa Q_A del contador. Éste es un *flip-flop* ÷2 simple. A continuación examinamos el funcionamiento de los *flip-flops* (Q_C, Q_B). Este contador de 2 bits es la clave del SN7492A. Estos dos bits forman un contador módulo 3, síncrono. Observe las ecuaciones lógicas de las entradas J y K para ambos *flip-flops*:

$$J_C = Q_B$$
$$K_C = 1$$
$$J_B = \overline{Q_C}$$
$$K_B = 1$$

Suponga que aplicamos un pulso *clear* [$R_{0(1)} = R_{0(2)} = 1$] para inicializar los dos *flip-flops* contadores, como se muestra en la figura 7.23d. Esta acción hace que (Q_C, Q_B) = (0, 0). Las entradas J y K de los *flip-flops* se convierten en (J_C, K_C) = (0, 1) y (J_B, K_B) = (1, 1). En el siguiente pulso de reloj aplicado a la entrada *Clock B*, *QC* se restablece a 0 lógico y *QB* alterna a 1 lógico, es decir, (Q_C, Q_B) = (0, 1). Después del pulso de reloj, las entradas J y K de los dos *flip-flops* cambian a las siguientes condiciones: (J_C, K_C) = (1, 1) y (J_B, K_B) = (1, 1). Estas condiciones de entrada indican a Q_C que alterne a 1 lógico y a Q_B que alterne a 0 lógico durante el siguiente pulso de reloj; así, el siguiente estado del contador de 2 bits será (Q_C, Q_B) = (1, 0). Este cambio de estado cambia de nuevo las entradas J y K sobre los dos *flip-flops*: (J_C, K_C) = (0, 1) y (J_B, K_B) = (0, 1). Estas condiciones de entrada indican a los dos *flip-flops* que realicen un *reset* durante el siguiente pulso de reloj, lo que lleva el contador a (Q_C, Q_B) = (0, 0), el estado inicial original. Así, la secuencia de estados en base 2 es (Q_C, Q_B) ⇒ (0, 0), (0, 1), (1, 0), (0, 0),... En base 10, la secuencia es 0, 1, 2, 0,...

La etapa más significativa del contador Q_D es otro *flip-flop* ÷2 simple. La figura 7.23b muestra el símbolo lógico para este módulo. A continuación describimos varias secuencias de conteo diferentes que pueden obtenerse con este módulo:

1. Conectamos Q_A a la entrada *Clock B* y proporcionamos una serie de pulsos de conteo a la entrada *Clock A*. El *flip-flop* Q_D será el bit más significativo, mientras que el *flip-flop* Q_A será el menos significativo. Generamos la siguiente secuencia de conteo: (Q_D, Q_C, Q_B, Q_A) = (0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 0,... Ver el diagrama de estados de la figura 7.23e.

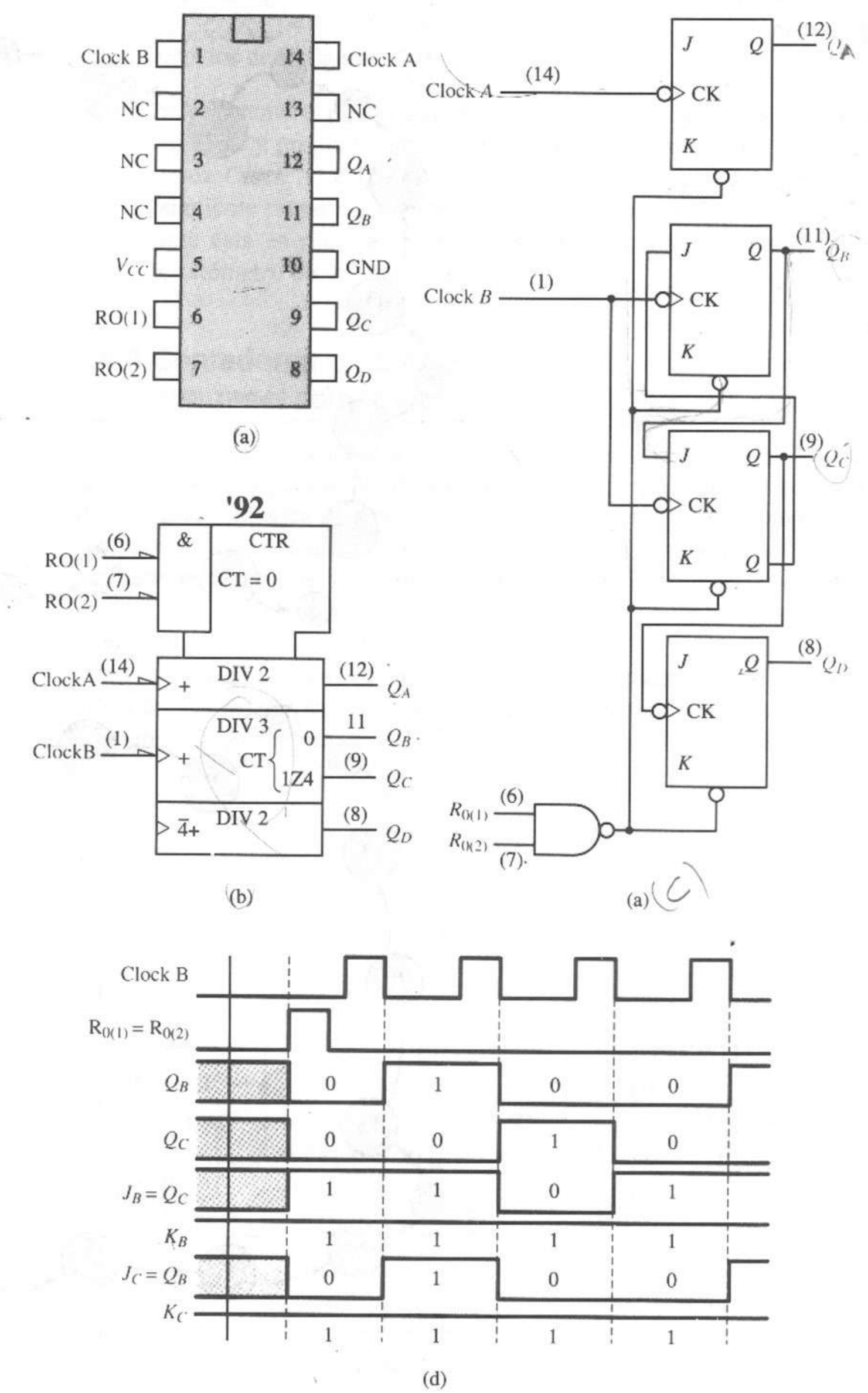
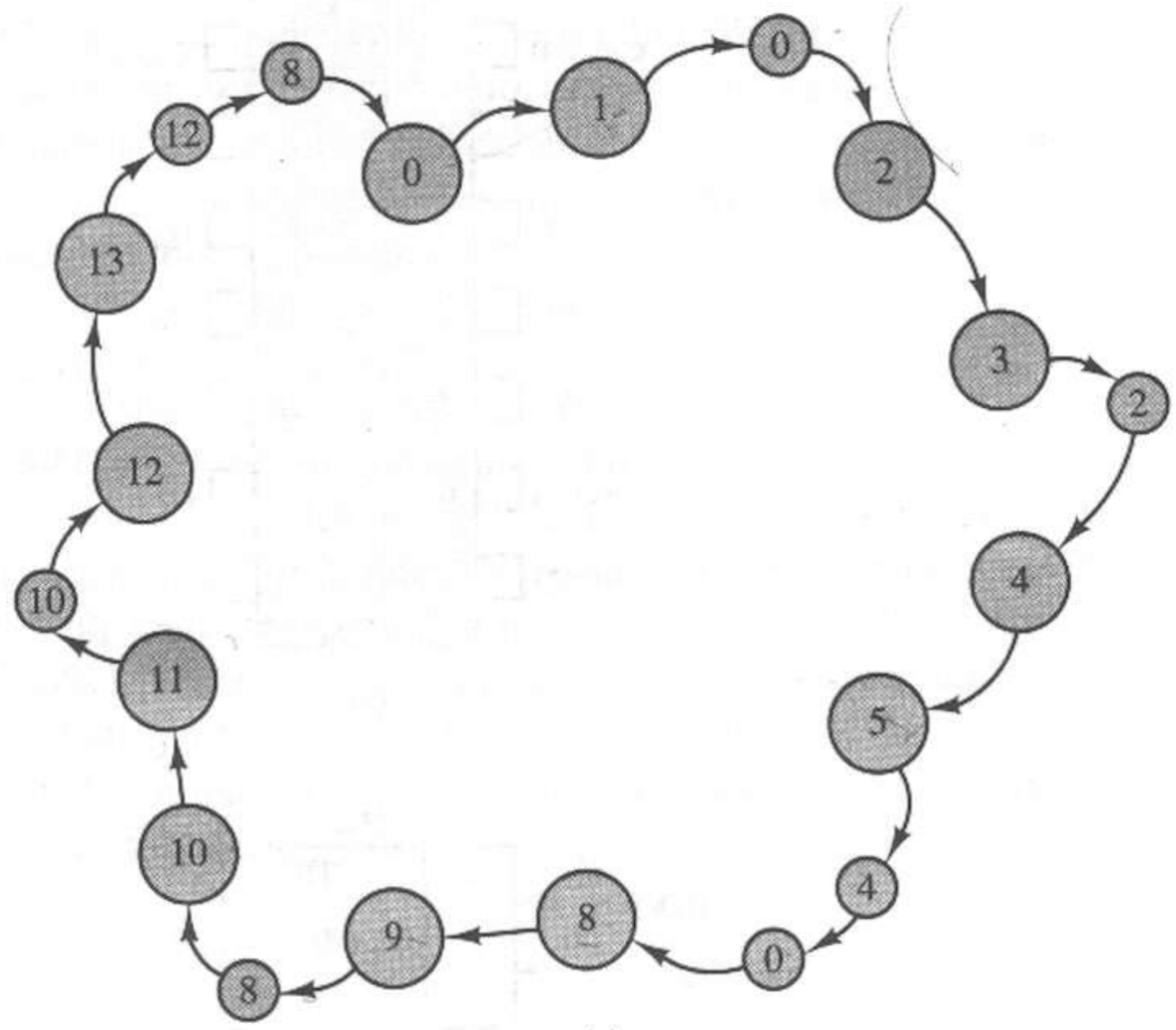
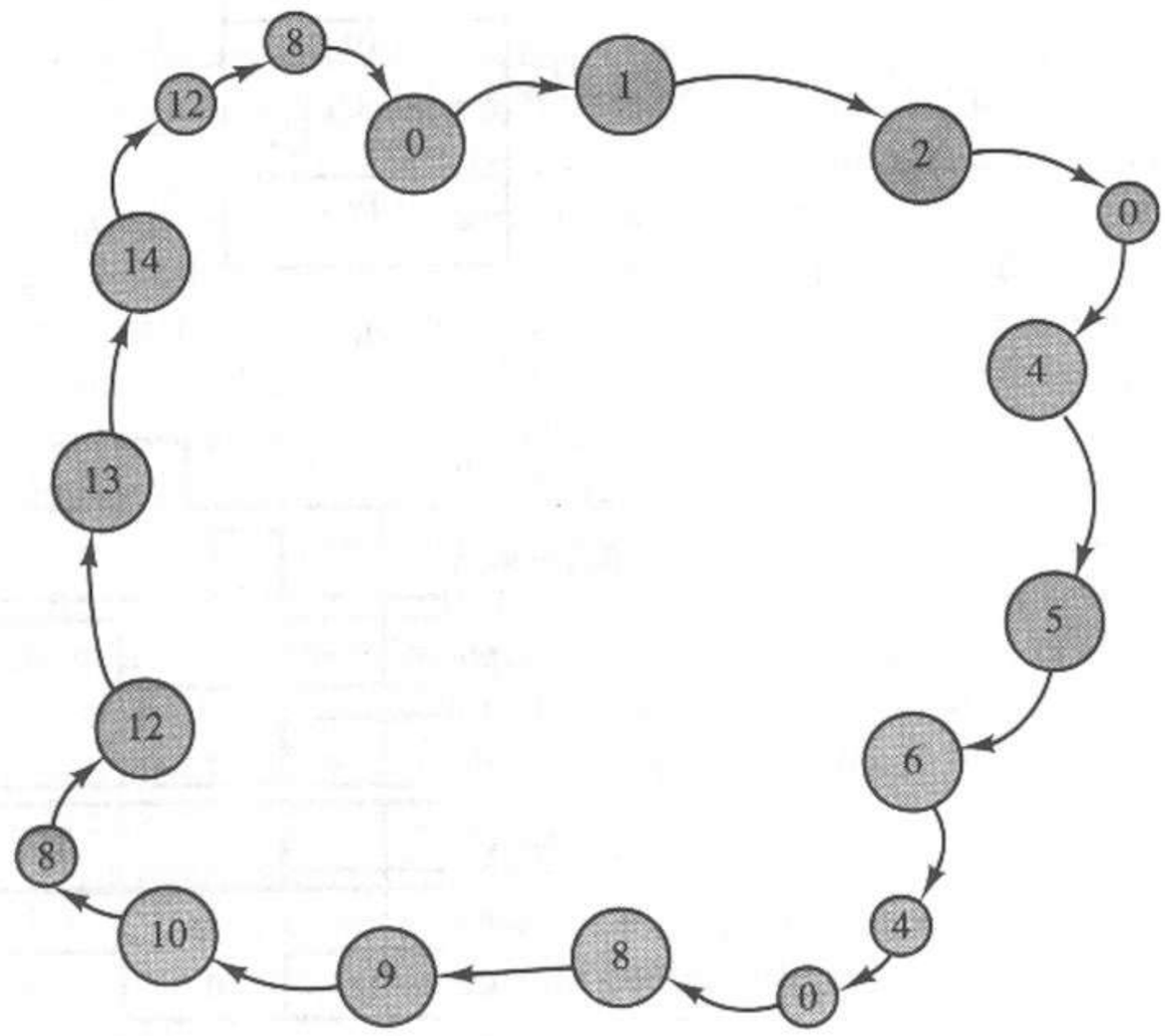


Figura 7.23 Contador asíncrono SN7492A. **(a)** *Pines* del paquete. **(b)** Símbolo lógico. **(c)** Diagrama lógico. **(d)** Diagrama de tiempos. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.



(e)



(f)

Figura 7.23 (continuación) Contador asíncrono SN7492A. **(e)** Diagrama de estados ($CKB = Q_A$). **(f)** Diagrama de estados ($CKA = Q_D$). *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

2. Conectamos la salida Q_D a la entrada *Clock A* y proporcionamos una secuencia de pulsos de conteo a la entrada *Clock B*. Q_A se convierte en el bit más significativo y generamos la siguiente sucesión de conteo: $(Q_A, Q_D, Q_C, Q_B) = 0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 0,...$ Ver el diagrama de estados de la figura 7.23f.

3. Si utilizamos el primer esquema de conexión (conectamos Q_A a la entrada *Clock B* y proporcionamos una secuencia de pulsos de conteo a la entrada *Clock A*) e ignoramos la etapa Q_D del contador, generamos la siguiente secuencia de conteo: $(Q_C, Q_B, Q_A) = 0,1,2,3,4,5,0,...$ Observe que ésta es precisamente la secuencia necesaria para el diseño de cronómetro de la figura 7.22.

7.4.4 Contadores módulo N con *reset* asíncrono

Podemos generalizar la técnica de diseño utilizada en el contador BCD asíncrono de la figura 7.20 para un contador módulo N general, como se muestra en la figura 7.24. La lógica de detección de estados consta de una compuerta AND con las entradas adecuadas para detectar el estado N, el módulo del contador. El número n de etapas de contador necesarias queda determinado por la relación

$$2^{n-1} < N < 2^n$$

Esta relación supone que N no es una potencia de 2, pues la red de retroalimentación no es necesaria en esos casos. El contador generalizado de la figura 7.24 requiere una señal de control *clear* común asíncrona. Los estados estables del

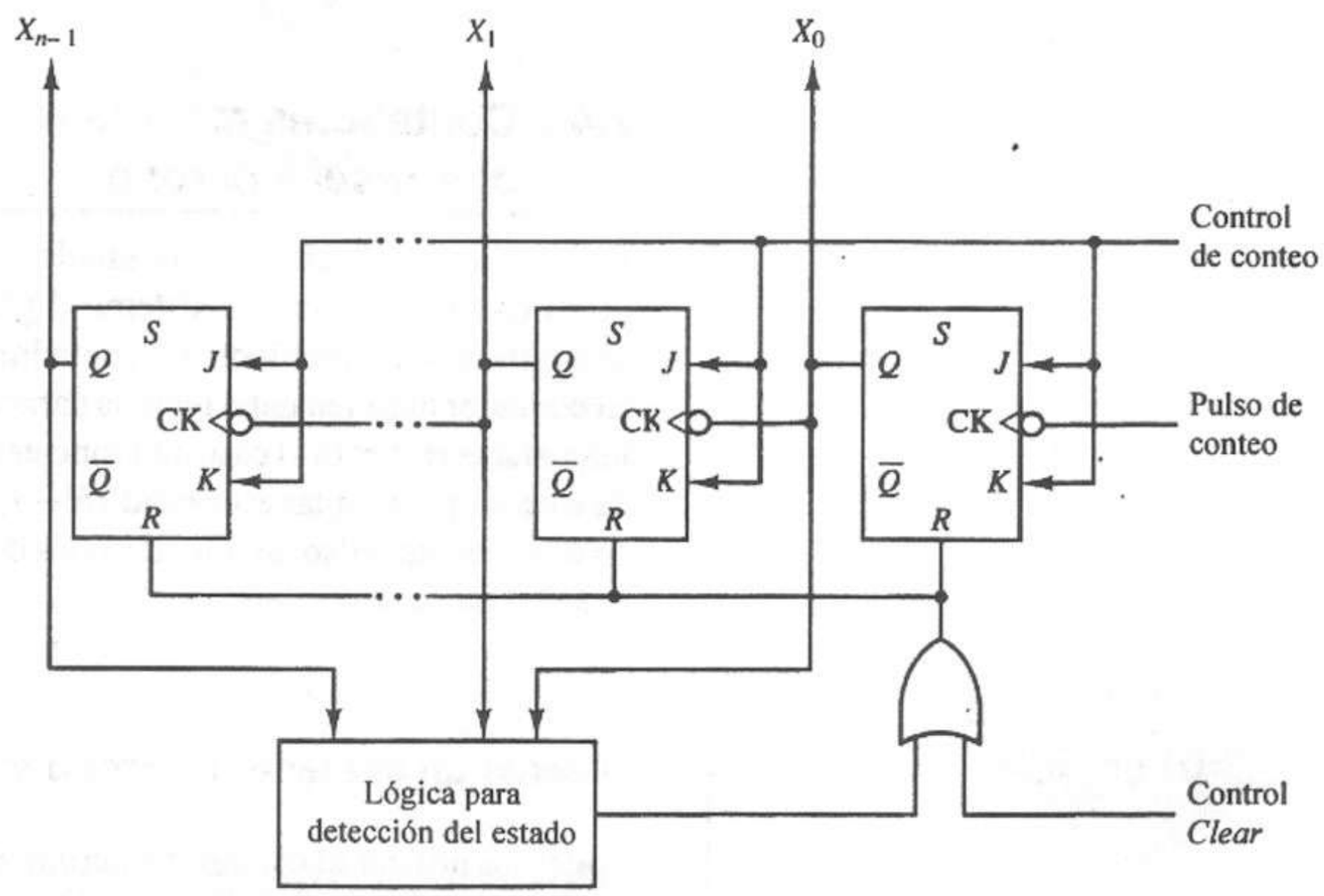


Figura 7.24 Contador asíncrono general módulo *N*.

contador serán 0, 1, 2,..., $N - 1$. La lógica de detección de estados captará el estado N y de inmediato hará que el contador pase del estado N al estado 0. Así, estos contadores con *reset* asíncrono siempre tendrán un estado transitorio N.

EJEMPLO 7.1

Diseñar un contador módulo 13 que utilice módulos de la serie SN7400. Como

$$2^3 < N = 13 < 2^4,$$

el número de etapas del contador debe ser $n = 4$.

Elegimos el contador binario asíncrono SN74293 de 4 bits y diseñamos un decodificador para el estado $N = 13$. El estado 13 representa el valor del contador

$$(Q_D, Q_C, Q_B, Q_A) = (1, 1, 0, 1)$$

En el funcionamiento normal, el contador pasará por los estados 0, 1,..., 12 antes de llegar al estado 13. Por tanto, podemos aprovechar el hecho de que el estado 13 será la primera ocurrencia de la condición lógica $Q_D = Q_C = Q_A = 1$ y podemos utilizar una compuerta AND de tres entradas para decodificar el estado 13, como se muestra en la figura 7.25a. Cuando ocurre el estado 13, la compuerta AND genera una señal de control *clear* y la aplica a las terminales *reset* $R_{0(1)}$ y $R_{0(2)}$ a través de la compuerta OR. Cuando el contador se estabiliza en el estado 0, la señal de control *reset* regresa a su condición inactiva, 0 lógico.

El diagrama de estados de la figura 7.25b muestra los estados de transición para este contador. Compare la figura 7.25b con la figura 7.14b. Observe que la lógica de detección de estados ha convertido el estado estable 13 de la figura 7.14b en un estado transitorio y ha evitado los demás estados, pasando en forma directa al estado 0.

7.4.5 Contadores módulo N con *reset* síncrono

En muchas aplicaciones, los estados transitorios de la figura 7.25b provocan picos (o *glitches*) en un sistema digital. Podemos eliminar este preocupante comportamiento mediante un contador síncrono en el circuito de la figura 7.24. El contador también debe tener la característica de *clear* síncrono. Una diferencia importante respecto al caso síncrono es que hay que diseñar la lógica de detección de estados para captar este estado $N - 1$, ya que debe hacerse un *reset* del contador en el *siguiente* pulso de reloj después de que la lógica de detección haya activado la señal de control *clear*.

EJEMPLO 7.2

Diseñar un contador síncrono módulo 13.

Podemos utilizar el contador síncrono de 4 bits SN74163 del circuito de la figura 7.26. Observe que la lógica de detección activará la entrada $\overline{Clear}$ al contador durante el estado 12, de modo que el *siguiente* estado después del estado 12 será el estado 0. Este circuito síncrono pasa por los estados estables de la figura 7.25b mientras elimina toda la actividad de los estados transitorios.

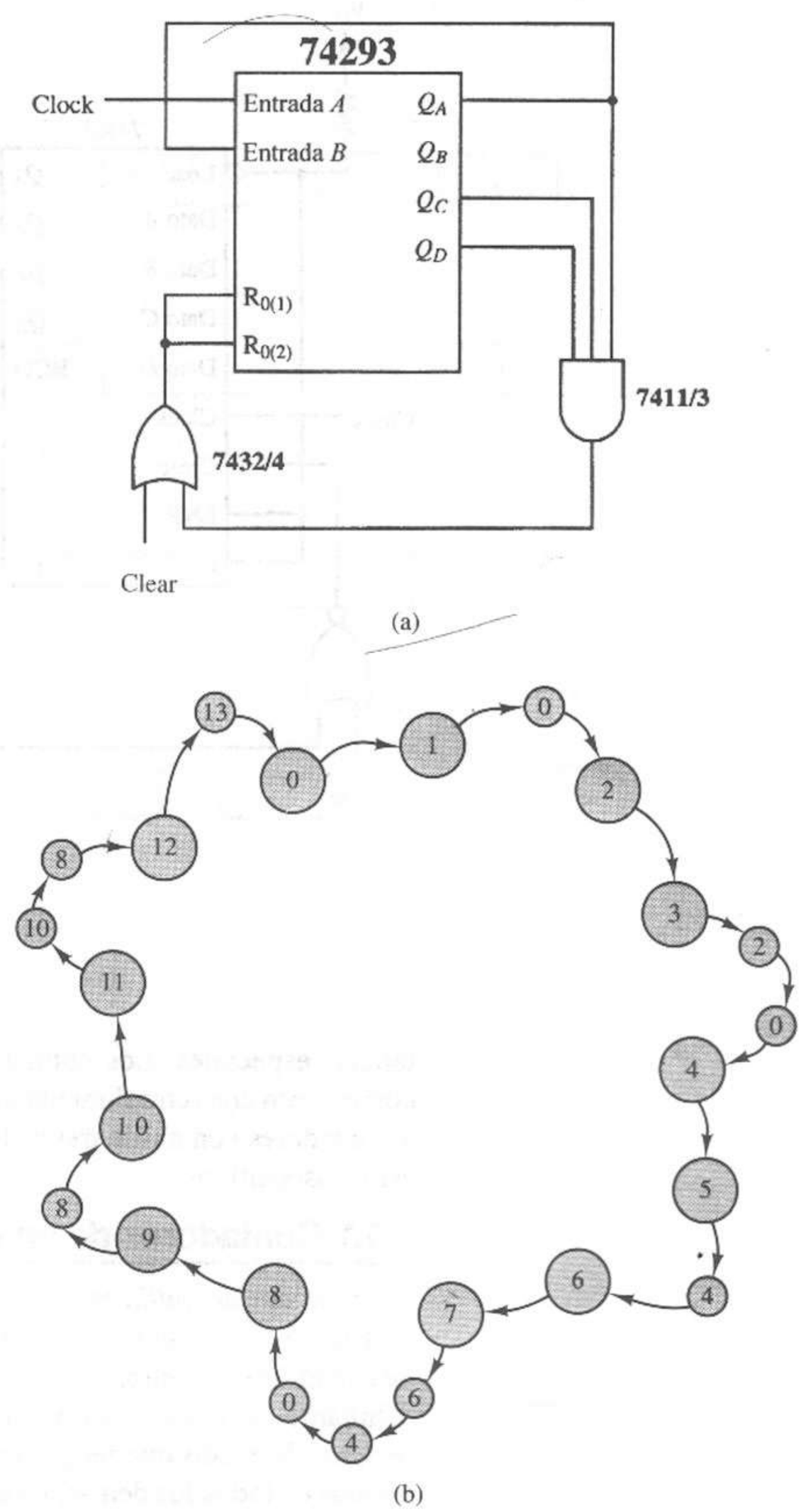


Figura 7.25 Contador módulo 13 con reset asíncrono. **(a)** Diagrama lógico. **(b)** Diagrama de estados.

7.5 Registros de corrimiento como contadores

Ahora examinaremos otra clase de contadores. Podemos utilizar los registros de corrimiento que hemos analizado en este capítulo como contadores en circuns-

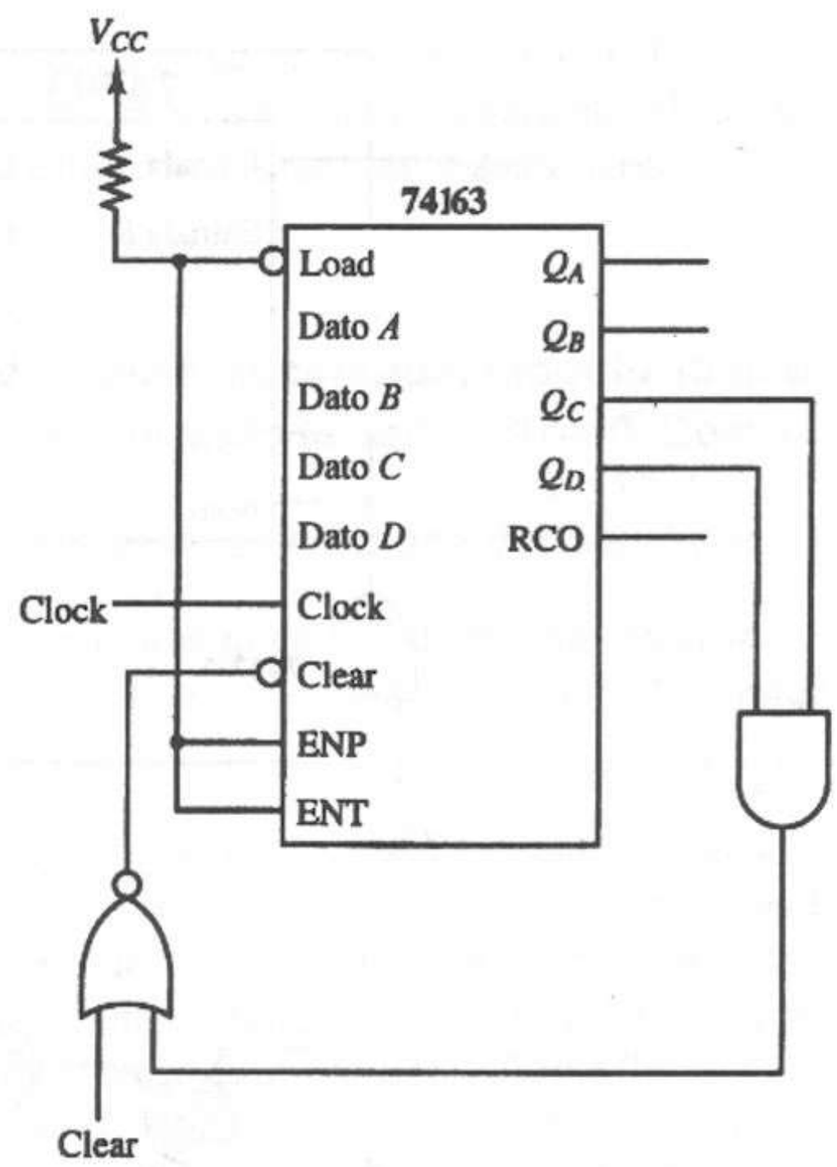


Figura 7.26 Contador módulo 13 con reset síncrono.

tancias especiales. Los contadores que se construyen mediante registros de corrimiento con retroalimentación se llaman por lo general contadores de anillo y contadores con anillo trenzado. Ahora investigaremos el comportamiento de estos dispositivos.

7.5.1 Contadores de anillo

Un *contador de anillo* es un dispositivo secuencial que tiene un *flip-flop* como registro de corrimiento por cada estado del contador. La salida serial del registro de corrimiento se retroalimenta al *pin* de entrada serial del registro. En el circuito resultante circula un patrón de bits alrededor del registro. Si inicializamos el registro de modo que tenga un único 1 lógico en su primer *flip-flop* y ceros lógicos en todos los demás, en el dispositivo circulará el 1 lógico en su ciclo de *flip-flops*, como se muestra en la figura 7.27. Sea n el número de *flip-flops* y, por tanto, el número de estados del contador. Los *flip-flops* del registro de corrimiento se rotulan $X_1, X_2, \ldots, X_n$. El funcionamiento del contador se inicia con un pulso en la línea de control *Initialize*. Esto hace que X_1 sea alta y $X_2, X_3, \ldots, X_{n-1}, X_n$ sean bajas. En este momento, sólo hay un 1 lógico en el *flip-flop* X_1. En la siguiente transición negativa de la señal de entrada *Clock*, el 1 lógico se transfiere del *flip-flop* X_1 al *flip-flop* X_2. El proceso continúa hasta que el 1 lógico llega al final del registro de corrimiento, el *flip-flop* X_n. En la transición negativa del siguiente pulso de reloj, el 1 lógico se transfiere mediante la línea de retroalimentación al primer *flip-flop* en el registro de corrimiento, X_1. Después, el proceso se repite. En otras palabras, el 1 lógico recorre un ciclo a través del registro de corrimiento cada n pulsos de reloj. Así, el contador de anillo tiene un único estado para cada

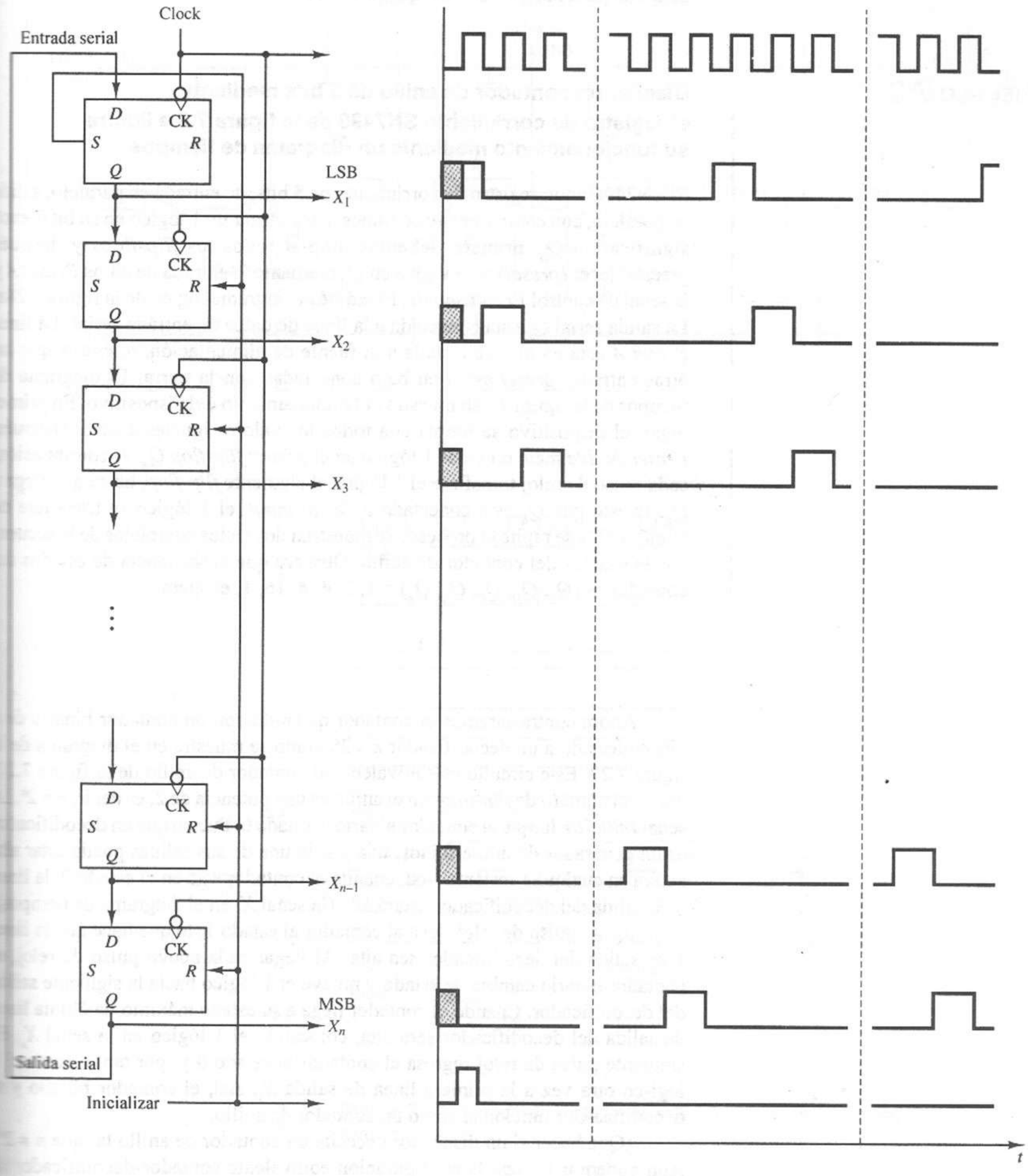


Figura 7.27 Contador de anillo

flip-flop. Podemos describir la secuencia de estados en valores decimales como $(X_n, X_{n-1},..., X_1)_2 = 1_{10}, 2_{10}, 4_{10}, 8_{10},..., (2^{n-1})_{10}$. Por ejemplo, un contador de anillo de 5 bits pasa por los estados 1, 2, 4, 8 y 16.

EJEMPLO 7.3

Diseñar un contador de anillo de 5 bits mediante el registro de corrimiento SN7496 de la figura 7.5 e ilustre su funcionamiento mediante un diagrama de tiempos

El SN7496 es un registro de corrimiento de 5 bits, de entrada en paralelo, salida en paralelo, con *clear* y *preset* comunes. Para cargar un 1 lógico en su bit menos significativo Q_A, primero debemos limpiar todos los *flip-flops* y después preestablecer (*preset*) un 1 lógico en Q_A mediante la entrada de datos *Preset A* y la señal de control *Preset enable*. Examine el diagrama lógico de la figura 7.28a. La salida serial Q_E está conectada a la línea de datos de entrada serial. La línea *Preset A* está en alto conectada a la fuente de alimentación, mientras que las otras entradas *preset* están en bajo conectadas con la tierra. El diagrama de tiempos de la figura 7.28b muestra el funcionamiento del dispositivo. En primer lugar, el dispositivo se limpia con todos los valores iguales a cero. Después, *Preset enable* mete un valor 1 lógico en el primer *flip-flop* Q_A. A continuación, cada pulso de reloj transfiere el 1 lógico al siguiente *flip-flop*, hasta que llega a Q_E. Puesto que Q_E está conectado a *Serial input*, el 1 lógico se transfiere de vuelta a Q_A y se repite el proceso. Se muestran dos ciclos completos de la secuencia de estados del contador de anillo. Observe que la secuencia de estados del contador es $(Q_E, Q_D, Q_C, Q_B, Q_A) = 1, 2, 4, 8, 16, 1$, etcétera.

Ahora contrastaremos el contador de anillo con un contador binario de k bits conectado a un decodificador k a 2^k, como se muestra en el diagrama de la figura 7.29. Este circuito es equivalente al contador de anillo de la figura 7.27, donde el número de *flip-flops* en el anillo es una potencia de 2, es decir, $n = 2^k$. La señal *Initialize* limpia el contador binario al estado 0. Puesto que un decodificador es un generador de mintérminos, una y sólo una de sus salidas puede estar alta activa en cualquier instante. Así, cuando el contador está en el estado 0, la línea 0 de salida del decodificador estará alta (la señal X_1 en el diagrama de tiempos). El siguiente pulso de reloj lleva al contador al estado 1, lo que hace que la línea 1 de salida del decodificador sea alta. Al llegar cada nuevo pulso de reloj, el contador binario cambia de estado y mueve el 1 lógico hacia la siguiente salida del decodificador. Cuando el contador llega a su estado máximo, la última línea de salida del decodificador será alta, colocando el 1 lógico en la señal X_n. El siguiente pulso de reloj regresa el contador al estado 0 y, por tanto, envía el 1 lógi-co otra vez a la primera línea de salida X_1. Así, el contador binario y el decodifica-dor funcionan como un contador de anillo.

¿Qué hacer si un diseñador necesita un contador de anillo tal que $n \neq 2^k$? Aún podemos utilizar la configuración equivalente contador-decodificador de la figura 7.29, reemplazando el contador binario con un contador módulo n. El decodificador debe satisfacer la relación

$$2^k > n > 2^{k-1}$$

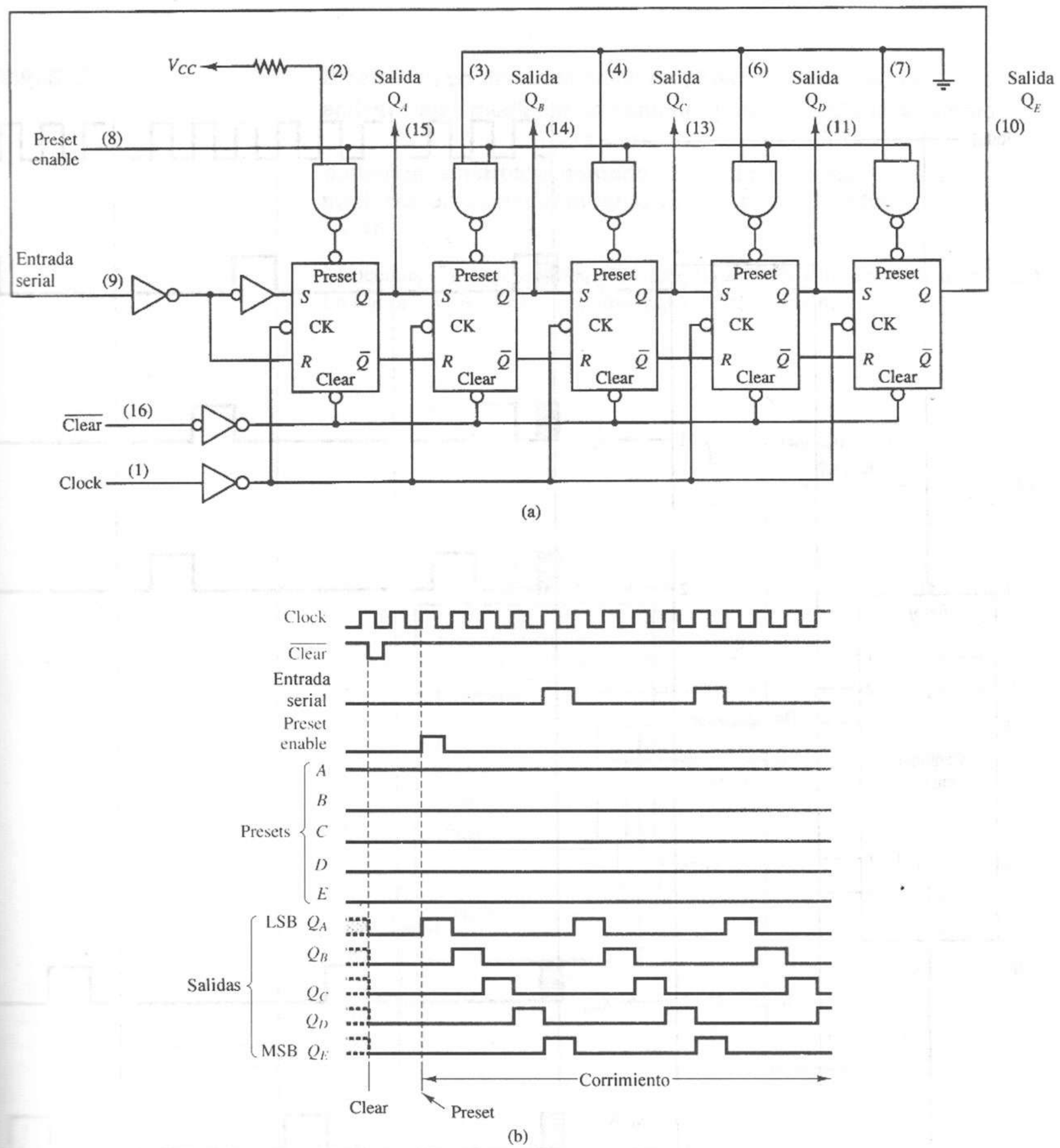


Figura 7.28 Ejemplo de contador de anillo. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

Utilizamos las salidas 0, 1, 2,..., $n-2$, $n-1$ del decodificador como las líneas de salida X_1, X_2, X_3,..., X_{n-1}, X_n del contador de anillo. No utilizaremos las líneas de salida n, $n+1$,..., 2^k-1 del decodificador.

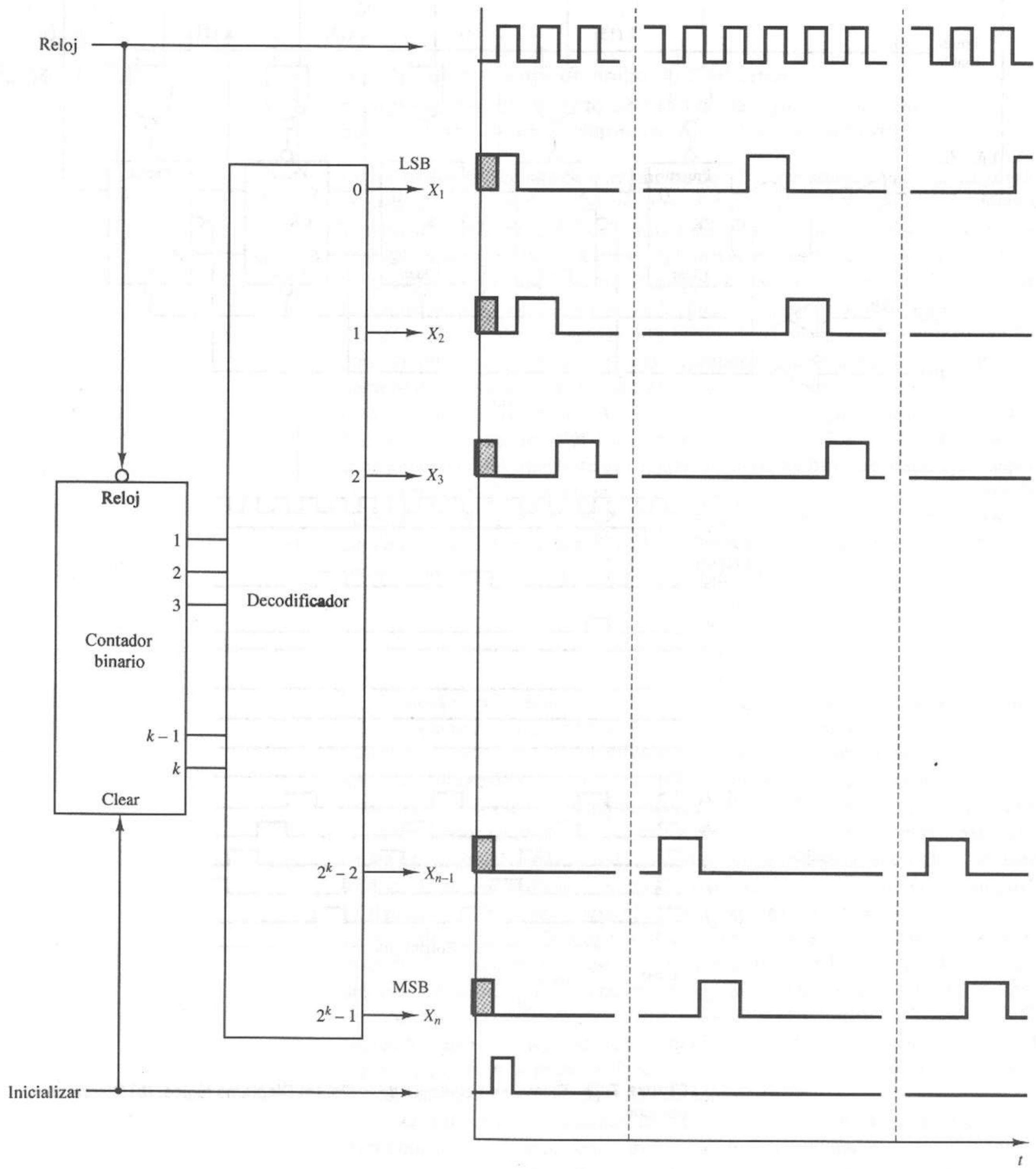


Figura 7.29 Equivalente del contador de anillo.

EJEMPLO 7.4

Diseñar un contador de anillo de 13 estados con salidas activas bajas mediante un contador y un decodificador. Como

$$2^4 > 13 > 2^3$$

podemos construir el contador de anillo deseado mediante un contador módulo 13 y un decodificador 4 a 16.

Utilicemos el contador módulo 13 de la figura 7.25 y un decodificador 74154. La figura 7.30 ilustra el diagrama lógico de este diseño.

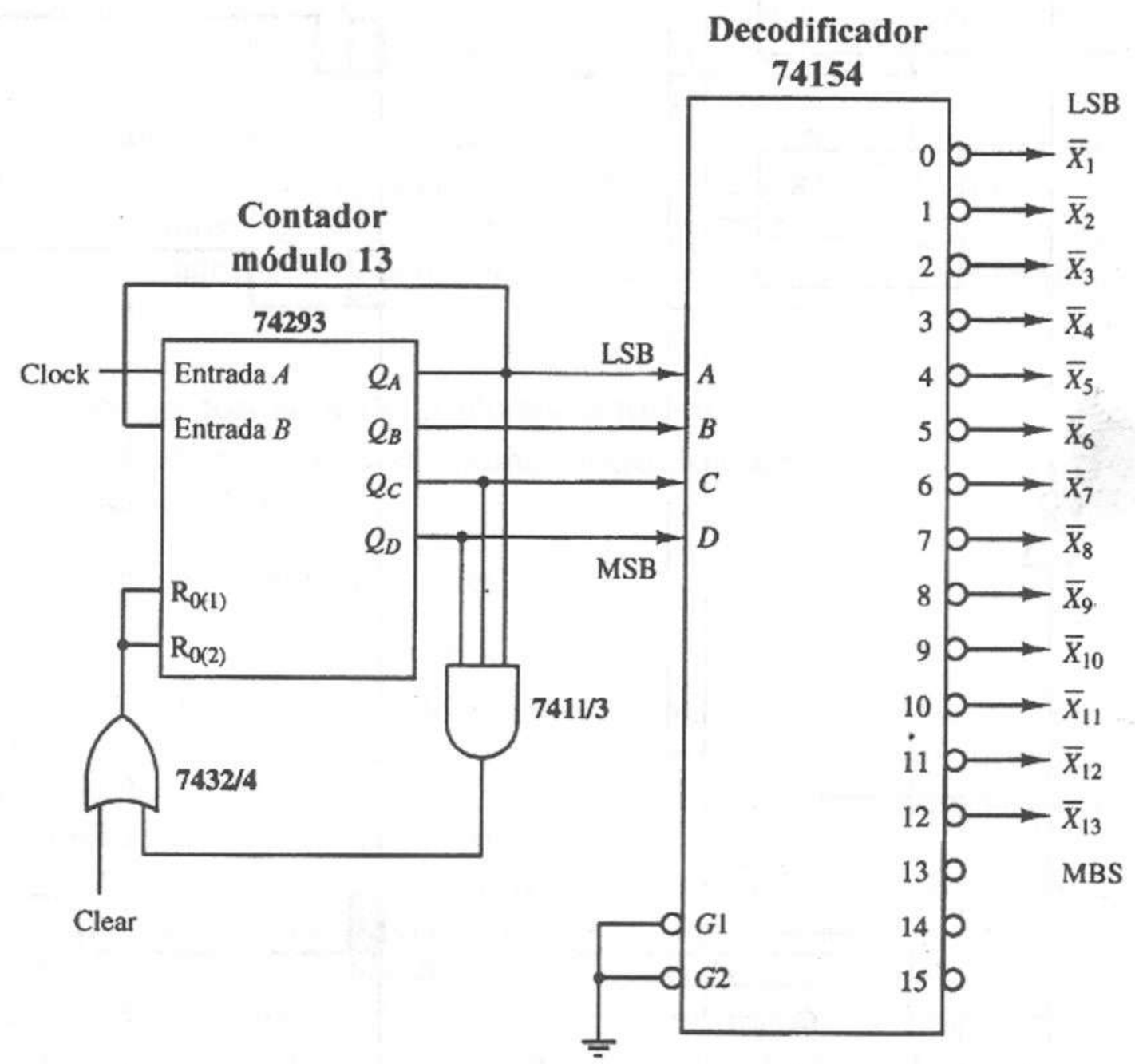


Figura 7.30 Equivalente al contador de anillo de trece estados.

7.5.2 Contadores de anillo trenzado

Un contador de anillo con una compuerta NOT en el ciclo de retroalimentación es un *contador de anillo trenzado*. A veces, este circuito se llama *contador de Johnson*. Observe el diagrama lógico de la figura 7.31. En este caso, hemos insertado una compuerta NOT entre el bit más significativo del contador, X_n, y

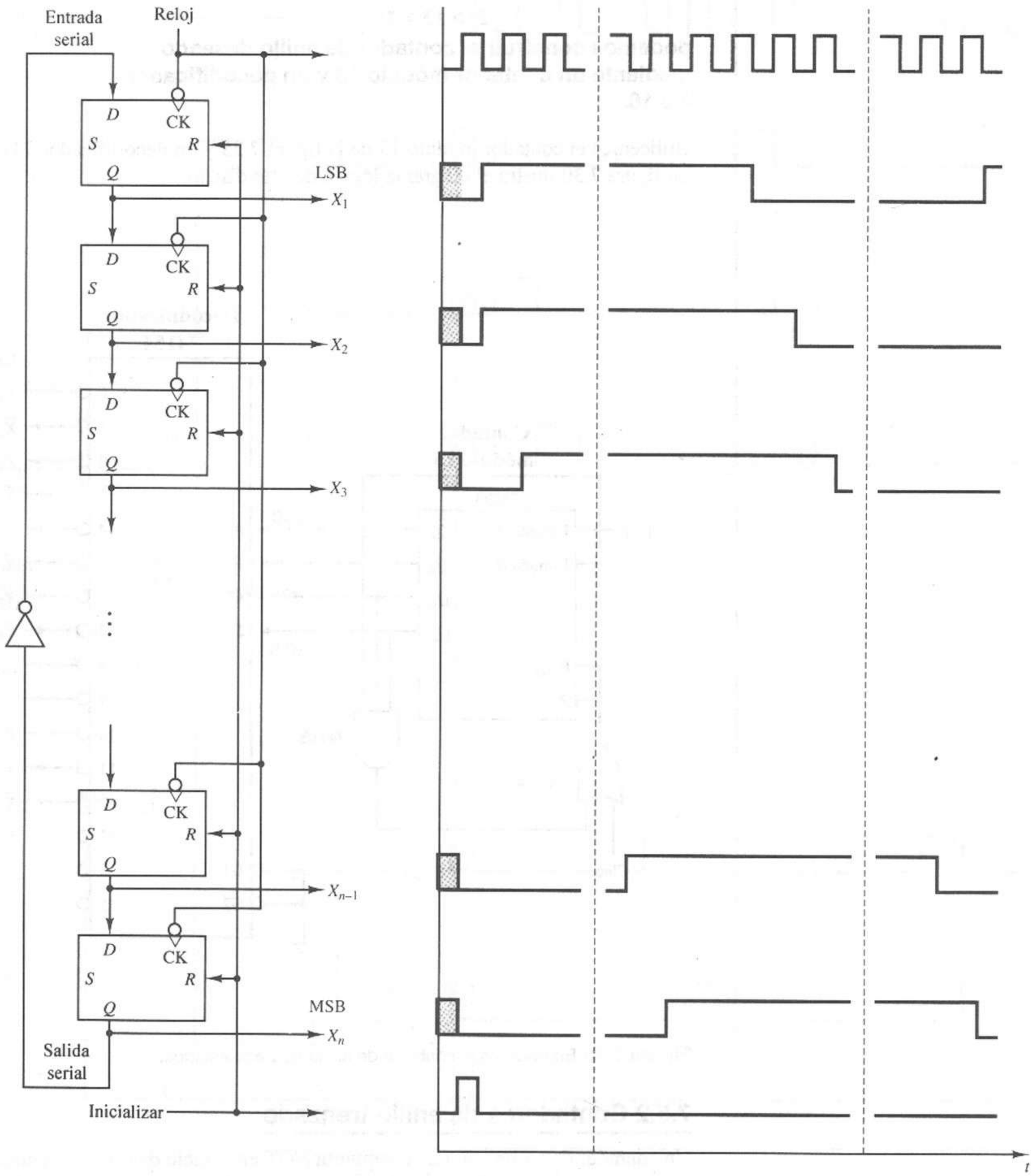


Figura 7.31 Contador de anillo trenzado.

la línea *Serial input* del registro de corrimiento. Además, observe que hemos alterado las conexiones de la señal *Initialize* para llevar el registro de corrimiento al estado nulo.

Examinemos el funcionamiento del registro de corrimiento en esta configuración. En primer lugar, la señal *Initialize* limpia el registro. Por tanto, la compuerta NOT proporcionará en un principio un 1 lógico en *Serial input* del registro de corrimiento (a la entrada *D* del primer *flip-flop* del registro de corrimiento). De hecho, esta retroalimentación del 1 lógico continúa hasta que X_n sea alta en un punto posterior de la secuencia de conteo. Así, al aplicar cada pulso a la entrada *Clock* del registro de corrimiento, el 1 lógico se mueve una posición hacia abajo en el registro (como un frente de onda) hasta que llega al último *flip-flop* del registro de corrimiento, X_n. Cuando el 1 lógico llega a X_n, la señal de retroalimentación en *Serial input* cambia a 0 lógico, de modo que ahora se mueve un frente de onda 0 lógico hacia abajo en el registro, al aplicarse los demás pulsos de reloj. El diagrama de tiempos de la figura 7.31 muestra un ciclo completo del contador de anillo trenzado. Cada señal de salida X_i ($i = 1, n$) es una onda cuadrada y se retarda con respecto a sus vecinas un periodo de tiempo de la señal de reloj.

¿Cuántos estados únicos posee este contador? Puesto que tarda n pulsos de reloj en propagar el 1 lógico hacia abajo en el registro y después otros n pulsos de reloj en regresar el registro al estado nulo, el contador de anillo trenzado tiene $2n$ estados únicos, donde n es el número de *flip-flops* del registro de corrimiento.

EJEMPLO 7.5

Diseñar un contador de anillo trenzado que tenga 10 estados únicos, usando módulos lógicos de la serie SN7400.

El número de *flip-flops* necesarios será

$$n = \frac{10}{2} = 5$$

Elegimos el SN7496 y colocamos una compuerta NOT en el ciclo de retroalimentación, como se muestra en la figura 7.32a. Aunque no se utilizan las señales *Preset i* ($i = A, B,..., E$), éstas se conectan a tierra para aumentar la inmunidad de la implantación respecto al ruido.

El diagrama de tiempos ilustra el uso adecuado del circuito. Una señal $\overline{Clear}$ activa baja inicializa el registro en el estado 0, forzando un 1 lógico en la señal *Serial input*. En la transición positiva de cada pulso de reloj, el 1 lógico se mueve de izquierda a derecha a través del registro, hasta que Q_E es alta. Entonces, la compuerta NOT mete un 0 lógico en el registro en los siguientes cinco pulsos de reloj, para regresar el registro al estado 0. Al aplicarse 10 pulsos de reloj, el contador de anillo trenzado pasa por sus estados únicos.

Podemos decodificar los estados del contador de anillo trenzado mediante compuertas AND, como se muestra en la figura 7.33. Las señales de salida del contador de anillo trenzado se enumeran en forma tabular. En la columna Ecuaciones lógicas del decodificador de estado aparece una única condición lógica para cada estado. Por ejemplo, observe la tercera fila de la tabla. Cuando $X_3 = 0$ y $X_2 = 1$, este estado del contador de anillo trenzado queda indicado de manera única. Revise las columnas X_3 y X_2 para verificar que ninguna otra fila (o estado del contador) tiene este patrón de bits para X_3 y X_2. También pueden

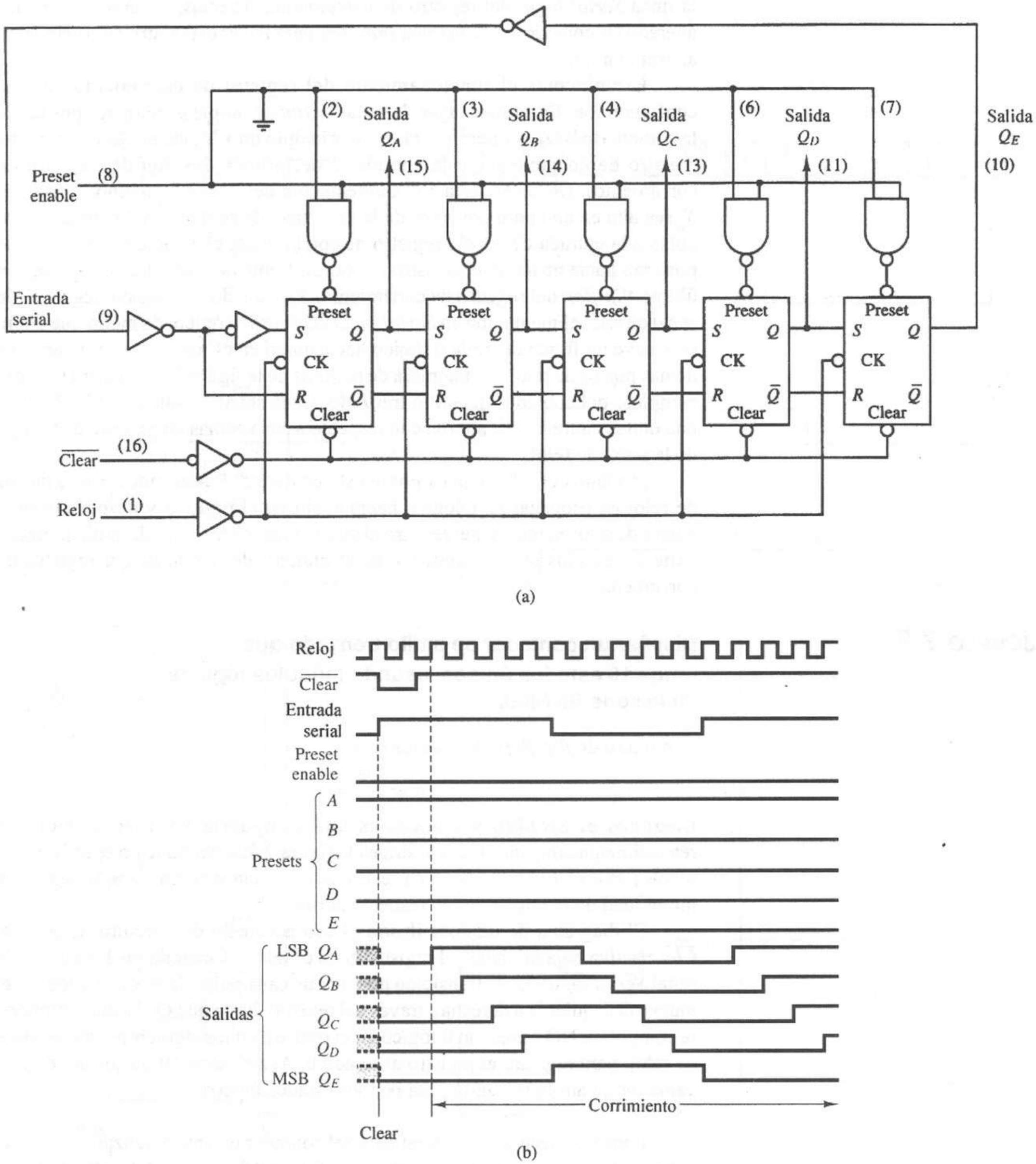


Figura 7.32 Contador de anillo trenzado de 10 estados. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

Señales de salida del contador de anillo trenzado								Ecuaciones lógicas del decodificador de estados
X_n	X_{n-1}	X_{n-2}	X_{n-3}	...	X_3	X_2	X_1	
0	0	0	0		0	0	0	$\overline{X}_n \bullet \overline{X}_1$
0	0	0	0		0	0	1	$\overline{X}_2 \bullet X_1$
0	0	0	0		0	1	1	$\overline{X}_3 \bullet X_2$
0	0	0	0		1	1	1	$\overline{X}_4 \bullet X_3$
				⋮				
0	0	0	0		1	1	1	$\overline{X}_{n-3} \bullet X_{n-4}$
0	0	0	1		1	1	1	$\overline{X}_{n-2} \bullet X_{n-3}$
0	0	1	1		1	1	1	$\overline{X}_{n-1} \bullet X_{n-2}$
0	1	1	1		1	1	1	$\overline{X}_n \bullet X_{n-1}$
1	1	1	1		1	1	1	$X_n \bullet \overline{X}_1$
1	1	1	1		1	1	0	$X_2 \bullet \overline{X}_1$
1	1	1	1		1	0	0	$X_3 \bullet \overline{X}_2$
1	1	1	1		0	0	0	$X_4 \bullet \overline{X}_3$
				⋮				
1	1	1	1		0	0	0	$X_{n-3} \bullet \overline{X}_{n-4}$
1	1	1	0		0	0	0	$X_{n-2} \bullet \overline{X}_{n-3}$
1	1	0	0		0	0	0	$X_{n-1} \bullet \overline{X}_{n-2}$
1	0	0	0		0	0	0	$X_n \bullet \overline{X}_{n-1}$
0	0	0	0		0	0	0	$\overline{X}_n \bullet \overline{X}_1$
0	0	0	0		0	0	1	$\overline{X}_2 \bullet X_1$
0	0	0	0		0	1	1	$\overline{X}_3 \bullet X_2$

Figura 7.33 Decodificación de estados para un contador de anillo trenzado.

identificarse de manera única todas las filas restantes, utilizando sólo dos señales lógicas adecuadas, como lo indica la última columna de la tabla. Si el contador de anillo trenzado tiene señales de salida reales y complementadas, podemos utilizar una compuerta AND de dos entradas para decodificar cada estado de salida. Si no se dispone de ambas, se necesitarán compuertas NOT.

EJEMPLO 7.6

Diseñar un generador de señales de tiempo mediante un contador de anillo trenzado y una lógica de decodificación de estados, con las siguientes especificaciones:

1. Las señales de tiempo serán altas activas durante un periodo de la señal de reloj (suben en la transición positiva de un pulso de reloj y bajan durante la transición positiva del siguiente pulso de reloj).
2. Se utilizará una señal de inicialización para sincronizar el generador de tiempos con los demás componentes del sistema. La señal *Initialize* será alta activa.
3. La primera señal de control de tiempos (f_1) generará un pulso alto durante la transición inicial del segundo pulso de reloj aplicado después de la instrucción de inicialización.
4. La segunda señal de control de tiempos (f_2) generará un pulso alto durante la transición inicial del octavo pulso de reloj aplicado después de la instrucción de inicialización.
5. La tercera señal de control de tiempos (f_3) generará un pulso alto durante la transición inicial del undécimo pulso de reloj aplicado después de la instrucción de inicialización.
6. Las formas de onda del tiempo (f_1, f_2, f_3) se repetirán cada 16 pulsos de reloj.

En primer lugar, generaremos la secuencia de estados para un contador de anillo trenzado de 16 estados, pues la secuencia de señales debe repetirse cada 16 pulsos de reloj. Como un contador de anillo trenzado tiene $2n$ estados, donde n es el número de *flip-flops*, seleccionaremos el registro de corrimiento SN74164 de entrada serial, salida en paralelo de la figura 7.4 para nuestra implantación. Las salidas del registro de corrimiento son (Q_H, Q_G, Q_F, Q_E, Q_D, Q_C, Q_B, Q_A). Si utilizamos una compuerta NOT para retroalimentar el complemento de QH en las entradas seriales, podemos describir el comportamiento de estado del contador de anillo trenzado mediante la tabla de la figura 7.34a.

Cuando el registro de corrimiento se inicializa en el estado nulo, asume los valores de la primera fila de la tabla (llamada pulso 0 del reloj). El primer pulso de reloj lleva el registro a la segunda fila (llamada pulso 1 del reloj), y el segundo lo lleva a la tercera fila (llamada pulso 2 del reloj). Este estado del contador de anillo trenzado será decodificado como la señal de tiempos f_1, como se muestra en la última columna de la tabla. Podemos continuar este mismo procedimiento para identificar los estados del contador que serán decodificados como las señales de tiempos f_2 y f_3. La lógica de decodificación para estas tres señales de tiempo es

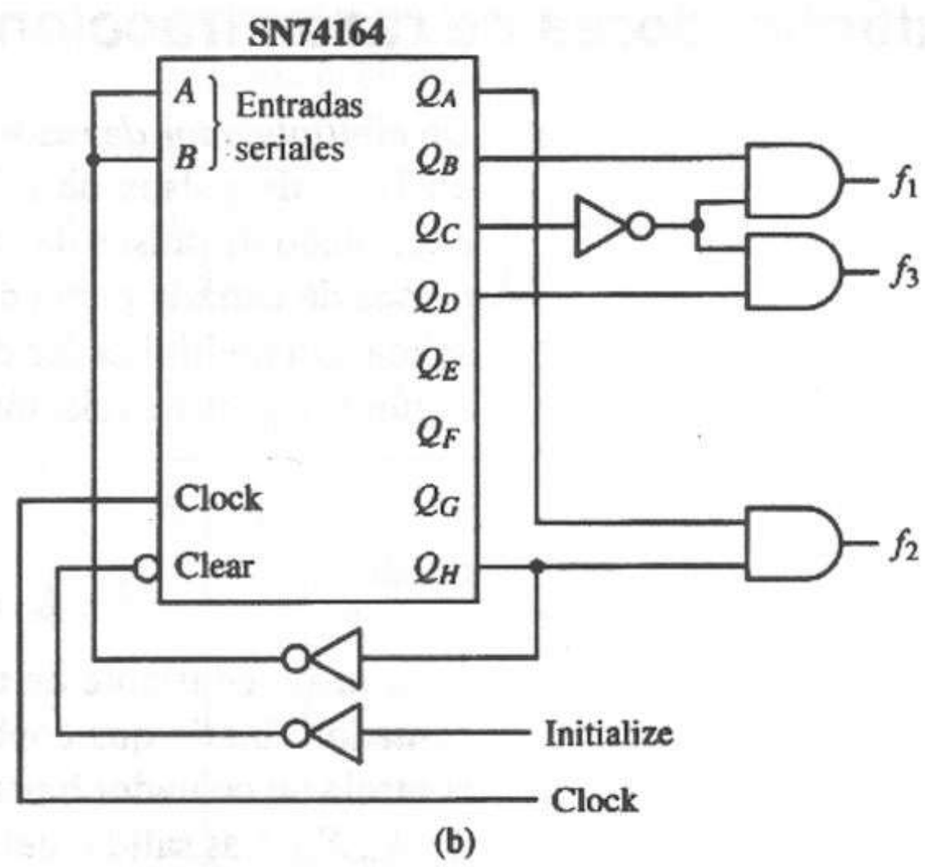
$$f_1 = \bar{Q}_C \cdot Q_B$$
$$f_2 = Q_H \cdot Q_A$$
$$f_3 = Q_D \cdot \bar{Q}_C$$

Estas señales se repiten cada 16 pulsos de reloj. Las figuras 7.34b y c muestran los diagramas lógicos de tiempos para este diseño.

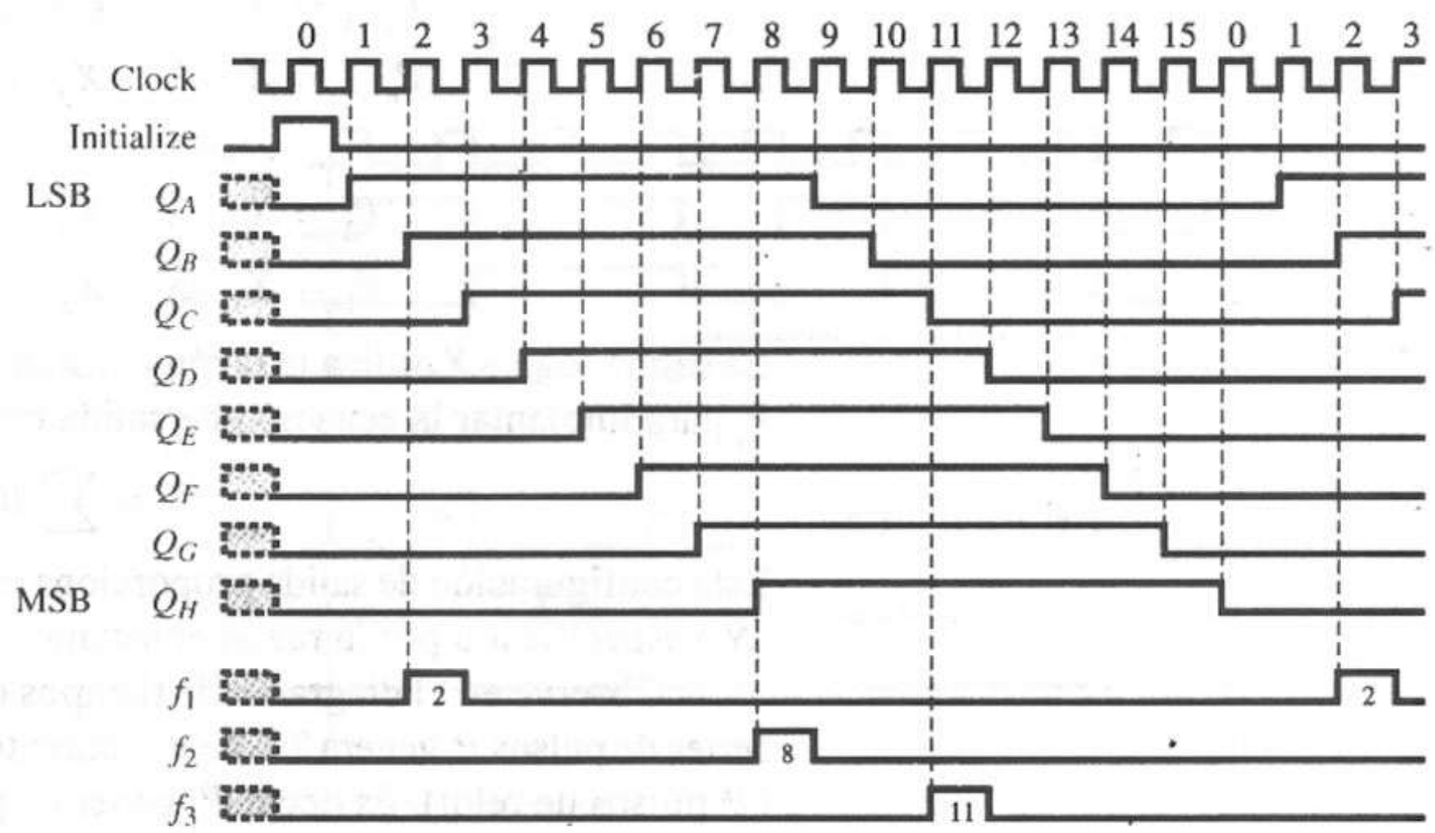
Pulso de reloj núm.	Estado (decimal)	Q_H	Q_G	Q_F	Q_E	Q_D	Q_C	Q_B	Q_A	Estado (decodificador)
0	0	0	0	0	0	0	0	0	0	$\bar{Q}_H \cdot \bar{Q}_A$
1	1	0	0	0	0	0	0	0	1	$\bar{Q}_B \cdot Q_A$
2	3	0	0	0	0	0	0	1	1	$f_1 = \bar{Q}_C \cdot Q_B$
3	7	0	0	0	0	0	1	1	1	$\bar{Q}_D \cdot Q_C$
4	15	0	0	0	0	1	1	1	1	$\bar{Q}_E \cdot Q_D$
5	31	0	0	0	1	1	1	1	1	$\bar{Q}_F \cdot Q_E$
6	63	0	0	1	1	1	1	1	1	$\bar{Q}_G \cdot Q_F$
7	127	0	1	1	1	1	1	1	1	$\bar{Q}_H \cdot Q_G$
8	255	1	1	1	1	1	1	1	1	$f_2 = Q_H \cdot Q_A$
9	254	1	1	1	1	1	1	1	0	$Q_B \cdot \bar{Q}_A$
10	252	1	1	1	1	1	1	0	0	$Q_C \cdot \bar{Q}_B$
11	248	1	1	1	1	1	0	0	0	$f_3 = Q_D \cdot \bar{Q}_C$
12	240	1	1	1	1	0	0	0	0	$Q_E \cdot \bar{Q}_D$
13	224	1	1	1	0	0	0	0	0	$Q_F \cdot \bar{Q}_E$
14	192	1	1	0	0	0	0	0	0	$Q_G \cdot \bar{Q}_F$
15	128	1	0	0	0	0	0	0	0	$Q_H \cdot \bar{Q}_G$
0	0	0	0	0	0	0	0	0	0	$\bar{Q}_H \cdot \bar{Q}_A$
1	1	0	0	0	0	0	0	0	1	$\bar{Q}_B \cdot Q_A$
2	3	0	0	0	0	0	0	1	1	$f_1 = \bar{Q}_C \cdot Q_B$

(a)

Figura 7.34 Ejemplo de contador de anillo trenzado de 16 estados. **(a)** Secuencia de estados.



(b)



(c)

Figura 7.34 (continuación) Ejemplo de contador de anillo trenzado con 16 estados. (b) Diagrama lógico. (c) Diagrama de tiempos.

7.6 Contadores de secuencia múltiple

En una sección anterior de este capítulo presentamos un tipo de contador de secuencia múltiple, el contador arriba/abajo. En ciertas ocasiones, un sistema digital requiere un contador con la capacidad de contar de varias formas, según lo indicado por algunas señales de control de entrada. En los capítulos 8 y 10 presentaremos algunos métodos de síntesis de circuitos secuenciales síncronos y asíncronos, respectivamente, que pueden contar en cualquier secuencia. Algunos de los ejemplos en esos capítulos demostrarán estos conceptos.

7.7 Multiplicadores de razón fraccionaria digital

Un *multiplicador de razón fraccionaria digital* es un dispositivo que transforma un flujo de pulsos de reloj de entrada (llamada entrada *Clock*) en un flujo controlado de pulsos de salida (llamada línea de salida Y). Sea N_i el número de pulsos de entrada para un periodo específico, y sea N_o el número de pulsos de salida. Un multiplicador de razón fraccionaria binaria produce pulsos de salida según la siguiente relación:

$$N_o = \frac{B}{2^n} N_i$$

donde

$$B = (B_{n-1}, B_{n-2}, \ldots, B_2, B_1, B_0)_2$$

es la razón constante de entrada al dispositivo y n es el número de etapas del contador binario que controla al módulo. Ver la figura 7.35. La entrada *Clock* controla un contador binario de n bits cuyas salidas se rotulan como ($X_n, X_{n-1}, \ldots, X_3, X_2, X_1$). Las salidas del contador se comparan mediante un AND con la señal *Clock* recibida para formar series de pulsos intermedios P_i ($i = 1, n$):

$$\begin{aligned}
P_n &= X_1 \cdot Clock \\
P_{n-1} &= \bar{X}_1 \cdot X_2 \cdot Clock \\
P_{n-2} &= \bar{X}_1 \cdot \bar{X}_2 \cdot X_3 \cdot Clock \\
&\ldots \\
P_2 &= \bar{X}_1 \cdot \bar{X}_2 \cdot \bar{X}_3 \cdot \ldots \cdot \bar{X}_{n-2} \cdot X_{n-1} \cdot Clock \\
P_1 &= \bar{X}_1 \cdot \bar{X}_2 \cdot \bar{X}_3 \cdot \ldots \cdot \bar{X}_{n-2} \cdot \bar{X}_{n-1} \cdot X_n \cdot Clock
\end{aligned}$$

La salida lógica Y utiliza la razón constante M y las señales de la serie de pulsos P_i para implantar la ecuación de salida como sigue:

$$Y = \sum B_{i-1} \cdot P_i$$

Esta configuración de salida proporciona el número adecuado de pulsos de salida (N_o) especificado por la razón constante B.

Observe en el diagrama de tiempos de la figura 7.35b que cada una de las series de pulsos P_i genera 2^{i-1} pulsos durante un periodo de secuencia del contador (2^n pulsos de reloj). Es decir, P_1 genera 1 pulso, P_2 genera 2 pulsos, P_3 genera 4 pulsos, P_4 genera 8 pulsos, etcétera. También es importante observar que estas series de pulsos no se traslapan en el tiempo. En consecuencia, podemos utilizar los bits de la razón constante B_i para llevar estas series de pulsos a la compuerta OR, como se muestra en el diagrama lógico.

Otra característica importante de estos dispositivos es la irregularidad del flujo de pulsos de salida. La separación entre los pulsos de la salida Y puede variar ampliamente. Los pulsos de salida se sincronizan con el reloj de entrada.

EJEMPLO 7.7

Considere la aplicación de un multiplicador de 3 bits de razón binaria. Sea $B = (7)_{10}$ la razón constante. En otras palabras, el multiplicador tendrá como salida siete pulsos por cada 8 pulsos del reloj de entrada. Uno de los pulsos de entrada será eliminado. ¿Cuál?

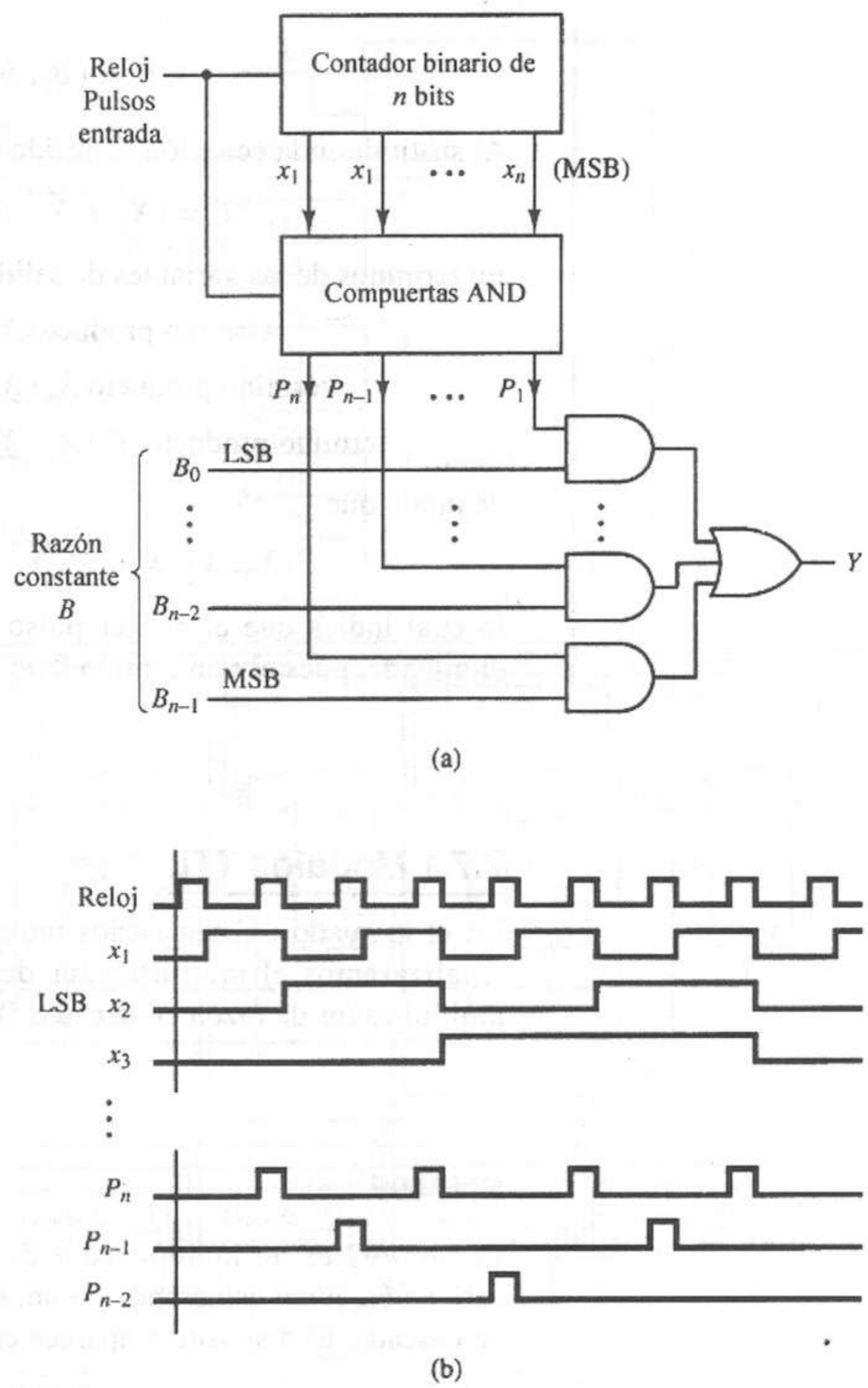


Figura 7.35 Multiplicador de razón fraccionaria binaria. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

Si examinamos la ecuación de salida para este dispositivo,

$$Y = B_2 \cdot P_3 + B_1 \cdot P_2 + B_0 \cdot P_1$$

donde

$$P_3 = X_1 \cdot Clock$$
$$P_2 = \bar{X}_1 \cdot X_2 \cdot Clock$$
$$P_1 = \bar{X}_1 \cdot \bar{X}_2 \cdot X_3 \cdot Clock$$

y

$$B = (B_2, B_1, B_0)_2 = (1, 1, 1)_2$$

Al sustituir en la ecuación de salida obtenemos

$$Y = (X_1 + \bar{X}_1 \cdot X_2 + \bar{X}_1 \cdot \bar{X}_2 \cdot X_3) \cdot Clock$$

En términos de las variables de salida del contador (X_3, X_2, X_1),

$$\text{término producto } X_1 \Rightarrow \sum m(1, 3, 5, 7)$$
$$\text{término producto } X_2 \cdot \underline{X}_1 \Rightarrow \sum m(2, 6)$$
$$\text{término producto } X_3 \cdot \underline{X}_2 \cdot \underline{X}_1 \Rightarrow \sum m(4)$$

de modo que

$$Y(X_3, X_2, X_1) = (\sum m(1, 2, 3, 4, 5, 6, 7)) \cdot Clock$$

lo cual indica que el primer pulso de reloj de cada secuencia de ocho será eliminado, pues el mintérmino 0 no aparece en la lista.

7.7.1 Módulos TTL

En el mercado existen varios multiplicadores de razón fraccionaria digital. Analizaremos el multiplicador de 6 bits de razón binaria (SN7497) y el multiplicador de razón en décadas (SN74167).

SN7497

El SN7497 es un multiplicador de 6 bits de razón fraccionaria binaria, con activación, *clear* asíncrono común, estroboscopio de salida y entradas y salidas en cascada. El dispositivo aparece en la figura 7.36. La razón de entrada es

$$B = (B5, B4, B3, B2, B1, B0)_2$$

y el número de pulsos de salida es

$$N_o = \frac{B}{2^6} N_i$$

La señal de control *Clear* es activa alta y opera en forma asíncrona para llevar todos los *flip-flops* del contador al estado nulo. La transición positiva de los pulsos de *Clock* causa la alternancia del contador. *Strobe* es una señal de habilitación activa alta para la razón de pulso de las salidas Y y Z.

Enable input y *Enable output* permiten formar una cascada con los módulos. Podemos construir un multiplicador con razón de 12 bits alimentando la *Enable output* del módulo menos significativo a *Enable input* y *Strobe* del módulo más significativo de la cascada. *Enable input* y *Enable output* son activas bajas.

Podemos utilizar *Unity/cascade input* para combinar las salidas de las unidades en cascada. La salida Z del módulo menos significativo se conecta a *Unity/cascade input* del más significativo, realizando un OR con las series de

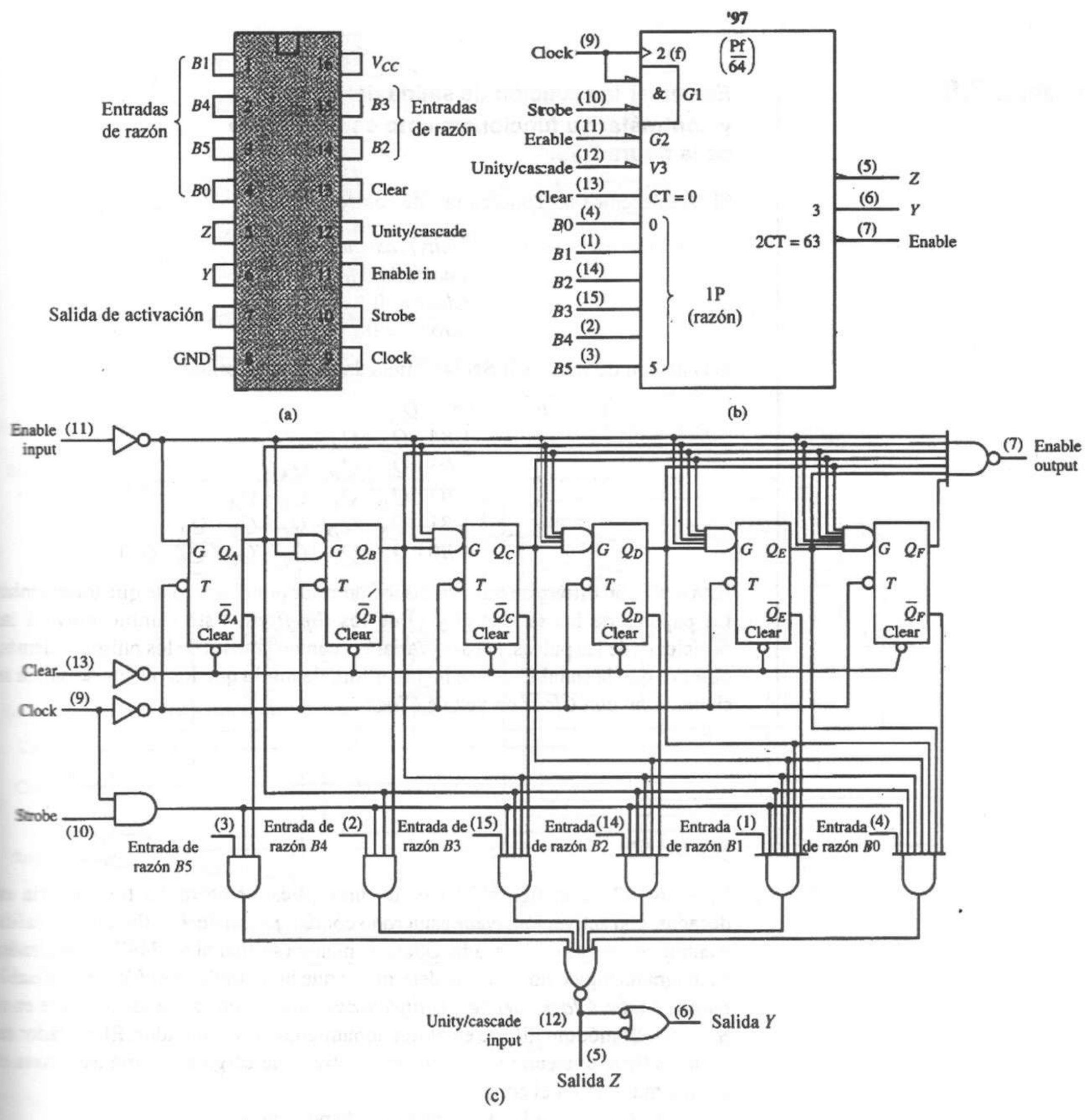


Figura 7.36 Multiplicador de seis bits de razón binaria, SN7497. **(a)** *Pines* del paquete. **(b)** Símbolo lógico. **(c)** Diagrama lógico. *Fuente*: The TTL Data Book Volumen 2, Texas Instruments Inc., 1985.

pulsos de los dos módulos. *Unity/cascade input* es alta si se utiliza un único módulo (haciendo $Z = \overline{Y}$).

EJEMPLO 7.8

Examinar la ecuación de salida del SN7497 y contrastar su funcionamiento con el diseño de la figura 7.35.

Si establecemos las condiciones de control

$$\begin{aligned} &Unity/cascade\ input = 1 \\ &Enable\ input = 0 \\ &Clear = 0 \\ &Strobe = 0 \end{aligned}$$

la ecuación de salida del SN7497 tiene la siguiente forma:

$$\begin{aligned} Y = \overline{Clock} \cdot \quad & (B5 \cdot \bar{Q}_A \\ & + B4 \cdot \bar{Q}_B \cdot Q_A \\ & + B3 \cdot \bar{Q}_C \cdot Q_B \cdot Q_A \\ & + B2 \cdot \bar{Q}_D \cdot Q_C \cdot Q_B \cdot Q_A \\ & + B1 \cdot \bar{Q}_E \cdot Q_D \cdot Q_C \cdot Q_B \cdot Q_A \\ & + B0 \cdot \bar{Q}_F \cdot Q_E \cdot Q_D \cdot Q_C \cdot Q_B \cdot Q_A) \end{aligned}$$

Esta ecuación difiere de nuestro diseño anterior por el hecho de que intercambia los papeles de las salidas Q y $\overline{Q}$ de los *flip-flops*. Este cambio moverá las posiciones de los pulsos, pero no variará el conteo general de los pulsos. Además, observe que la señal *Clock* se ha invertido, de modo que los pulsos de salida se sincronizan con $\overline{Clock}$ en vez de *Clock*.

SN74167

El SN74167 de la figura 7.37 es un multiplicador de razón fraccionaria en décadas, con activación, *clear* asíncrono común, *preset* de 9, activación de salida y entradas y salidas en cascada. Opera de manera similar al SN7497 ya analizado. El diagrama lógico nos permite determinar que las entradas *Enable input, Enable output, Clear, Clock, Strobe* y *Unity/cascade* operan en forma idéntica que en el SN7497. El módulo difiere en el funcionamiento de su contador. El contador de décadas tiene una entrada *Set to 9* alta activa que carga en forma asíncrona el conteo máximo en el contador.

La ecuación de la razón para este dispositivo es

$$N_o = \frac{B}{10} N_i$$

donde la razón constante $B = (B_3, B_2, B_1, B_0)_2$ está restringida a los valores $0 \leq B \leq 9$.

EJEMPLO 7.9

Determinar la ecuación lógica para la salida *Y* del SN74167 utilizando las mismas condiciones de control del ejemplo 7.8.

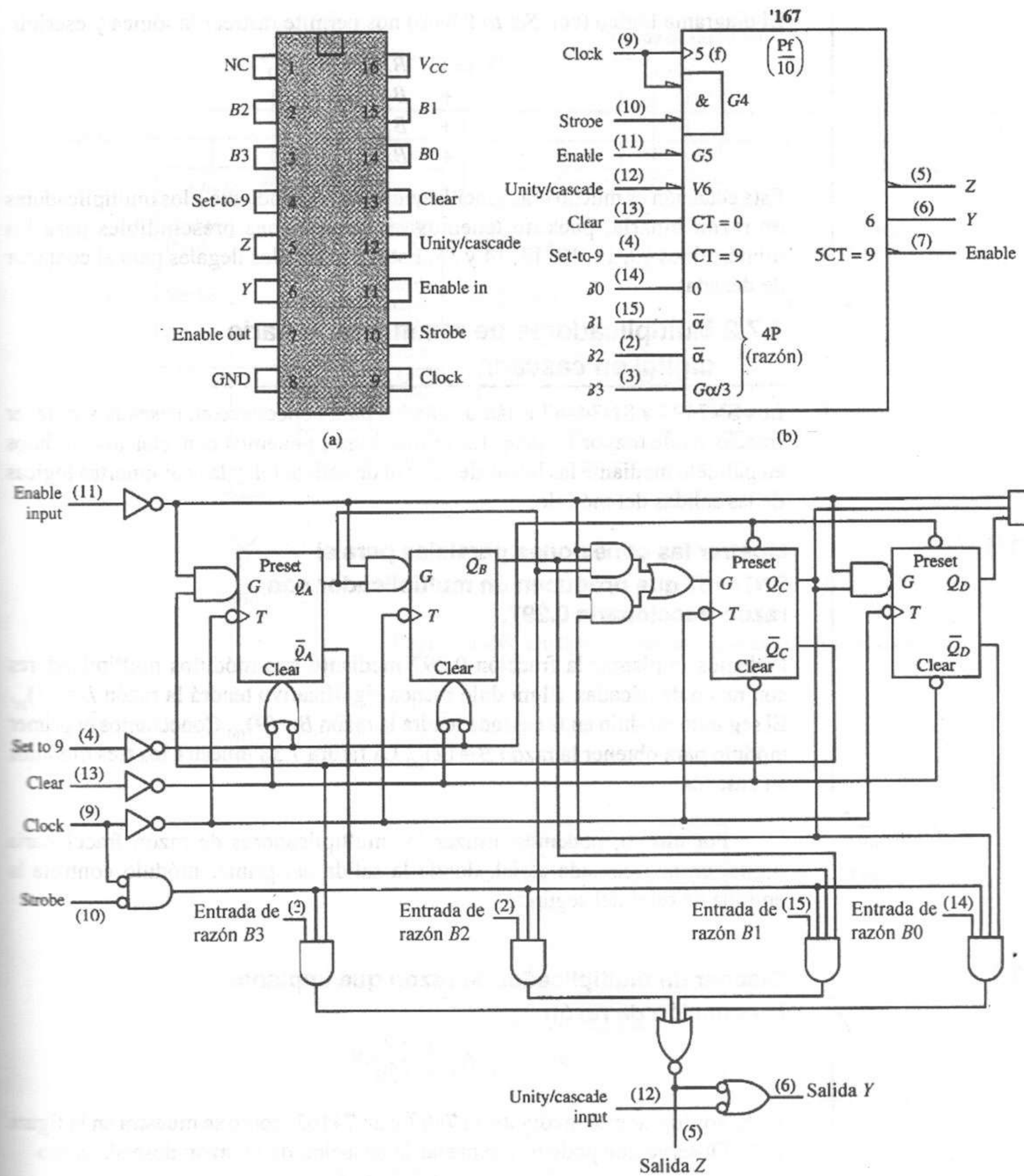


Figura 7.37 Multiplicador de razón de década, SN74167. **(a)** *Pines* del paquete. **(b)** Símbolo lógico. **(c)** Diagrama lógico.

El diagrama lógico (con *Set to 9* bajo) nos permite rastrear la lógica y escribir

$$\begin{aligned} Y = (\ & B_3 \cdot \bar{Q}_C \\ + \ & B_2 \cdot Q_A \\ + \ & B_1 \cdot Q_B \cdot \bar{Q}_A \\ + \ & B_0 \cdot \bar{Q}_D \cdot Q_C) \end{aligned}$$

Esta ecuación es mucho más sencilla que la correspondiente a los multiplicadores de razón binaria, pues no tenemos las condiciones prescindibles para los mintérminos 10, 11, 12, 13, 14 y 15. Éstos son estados ilegales para el contador de décadas.

7.7.2 Multiplicadores de razón fraccionaria digital en cascada

Los SN7497 y SN74167 están diseñados para conectarse en cascada y obtener fracciones de mayor tamaño. En primer lugar, podemos conectar los módulos en paralelo mediante las líneas de control de activación y las compuertas lógicas de las salidas del módulo.

EJEMPLO 7.10

Mostrar las conexiones paralelas para el SN74167 que producen un multiplicador con razón fraccionaria 0.297.

Podemos implantar la fracción 0.297 mediante tres módulos multiplicadores con razón de décadas. El módulo menos significativo tendrá la razón $B = (7)_{10}$. El segundo módulo en la cascada tendrá la razón $B = (9)_{10}$. Conectamos el primer módulo para obtener la razón $B = (2)_{10}$. La figura 7.38 muestra los tres módulos en cascada.

Por último, podemos utilizar los multiplicadores de razón fraccionaria digital en una cascada serial, donde la salida del primer módulo controla la entrada de reloj del segundo.

EJEMPLO 7.11

Diseñar un multiplicador de razón que implante la ecuación de razón

$$N_o = \frac{63}{320} N_i$$

Podemos lograr esto mediante un 7497 y un 74163, como se muestra en la figura 7.39. Observe que podemos expresar la ecuación de la razón deseada como

$$N_o = \left(\frac{7}{10}\right)\left(\frac{18}{64}\right) N_i$$

Así, $B = (0111)_2$ para el 74167 y $B = (010010)_2$ para el 7497.

7.8 Resumen

En este capítulo analizamos el diseño y el funcionamiento de varios módulos lógicos secuenciales estándar, como los registros de corrimiento y los contadores. Muchos de éstos están disponibles como funciones TTL estándar [1], o bien en las bibliotecas de diseño que se usan para crear circuitos VLSI, arreglos de

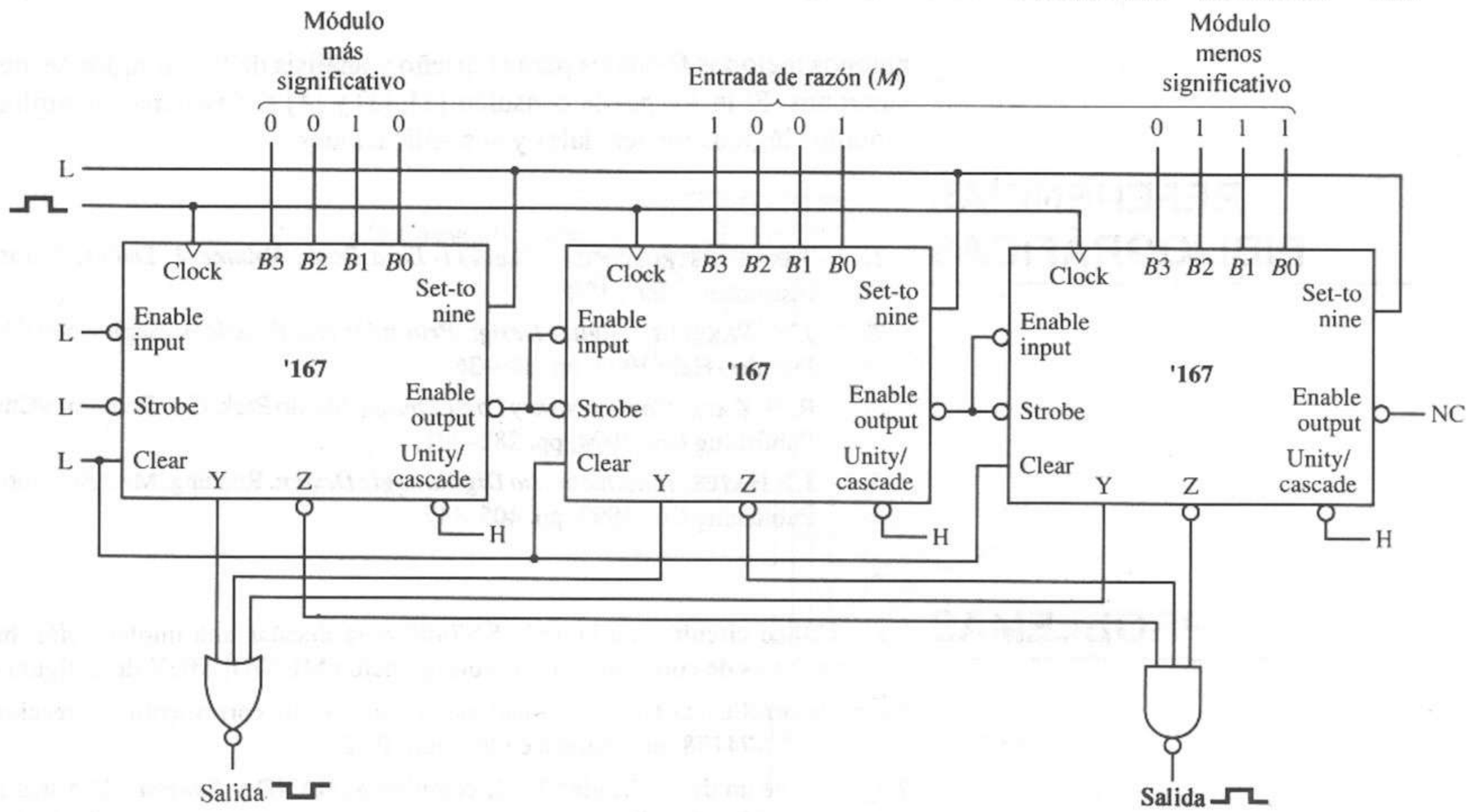


Figura 7.38 Multiplicadores de razón de década en cascada.

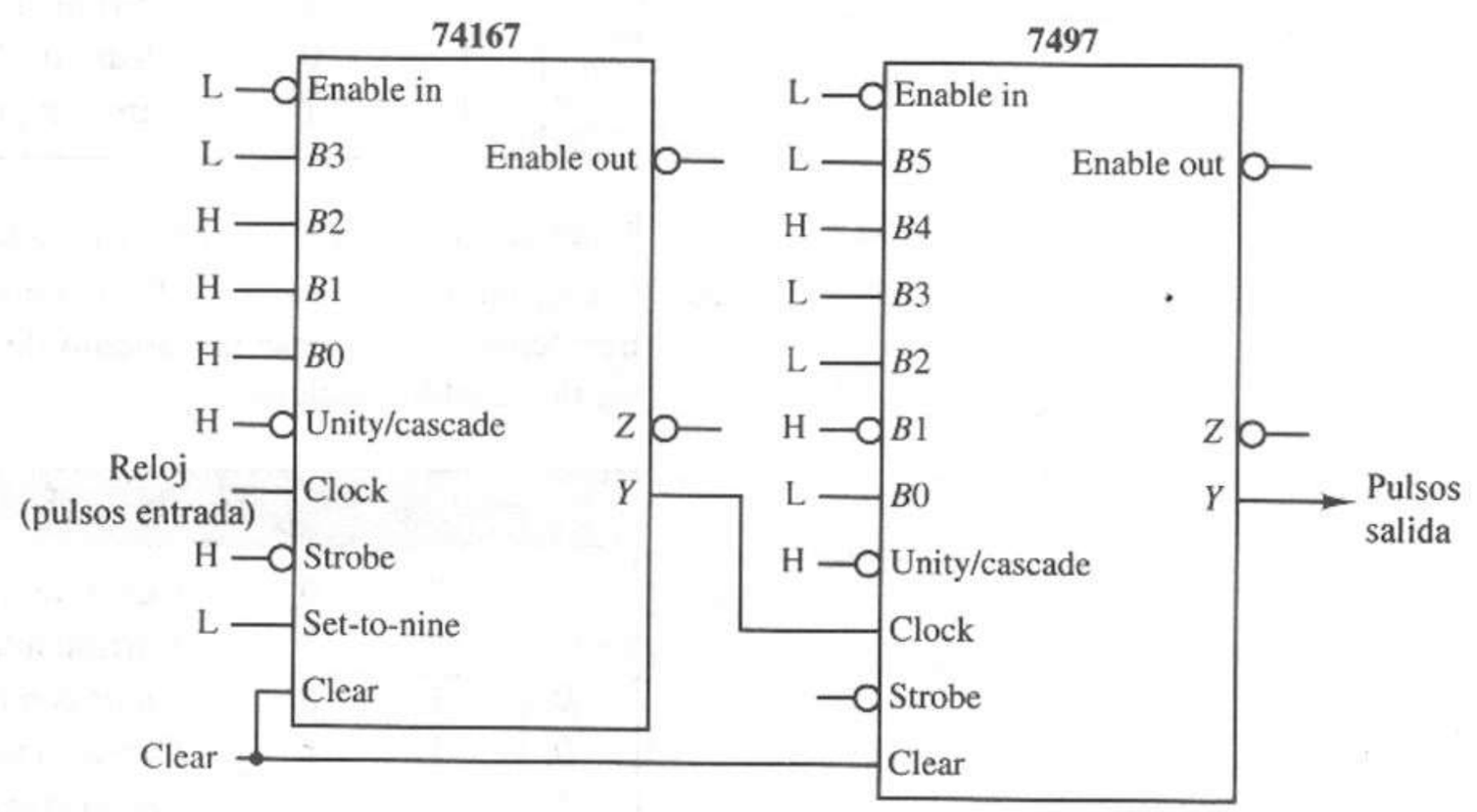


Figura 7.39 Multiplicadores de razón en cascada serial.

compuertas programables y tarjetas de circuitos impresos. Presentamos varios ejemplos a fin de ilustrar el uso de estos módulos como bloques de construcción para crear circuitos mayores. Con frecuencia, es mejor utilizar estos módulos en un diseño en vez de crear circuitos adaptados. En el capítulo 8 presentaremos

algunos métodos formales para el diseño y análisis de los circuitos secuenciales síncronos. El lector puede consultar [2], [3] y [4] si desea más ejemplos de los módulos lógicos secuenciales y sus aplicaciones.

REFERENCIAS BIBLIOGRÁFICAS

1. TEXAS INSTRUMENTS, *The TTL Data Book, Volume 2*. Dallas, Texas: Texas Instruments, Inc., 1988.
2. J. F. WAKERLY, *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice-Hall, 1990, pp. 349–369.
3. R. H. KATZ, *Contemporary Logic Design*. Menlo Park, CA: Benjamin/Cummings Publishing Co., 1994, pp. 282–321.
4. J. P. HAYES, *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley Publishing Co., 1993, pp. 405–447.

PROBLEMAS

7.1 Utilice circuitos de la serie SN7400 para diseñar una implantación mediante registros de corrimiento de la configuración MUX/DEMUX de la figura 4.29.

7.2 Desarrolle una tabla funcional para el registro de corrimiento bidireccional de 8 bits SN74198 que aparece en la figura P7.2.

7.3 Utilice un decodificador 2 a 4, compuertas NAND y *flip-flops* D disparados por transición para diseñar un módulo de registro de corrimiento de 4 bits con la siguiente tabla funcional:

S1	S0	Modo
0	0	Corrimiento a la derecha (los cuatro bits)
0	1	Corrimiento a la izquierda (los cuatro bits)
1	0	*Clear* común síncrono
1	1	Carga en paralelo síncrona

Trace un diagrama lógico para su módulo.

7.4 Utilice un decodificador 3 a 8, compuertas NAND y *flip-flops* D disparados por transición para diseñar un módulo de registro de corrimiento de 4 bits con la siguiente tabla funcional:

S2	S1	S0	Modo
0	0	0	Corrimiento a la derecha (los cuatro bits)
0	0	1	Corrimiento a la izquierda (los cuatro bits)
0	1	0	*Clear* común síncrono
0	1	1	Carga en paralelo síncrona
1	0	0	*Preset* síncrono del bit más significativo en 1 y *clear* a los demás bits
1	0	1	Retención síncrona de datos
1	1	0	Contador de anillo (la salida Q del bit menos significativo se retroalimenta como entrada serial al bit más significativo)
1	1	1	Contador de anillo trenzado (la salida $\overline{Q}$ del bit menos significativo se retroalimenta como entrada serial al bit más significativo)

Trace un diagrama lógico para su módulo.

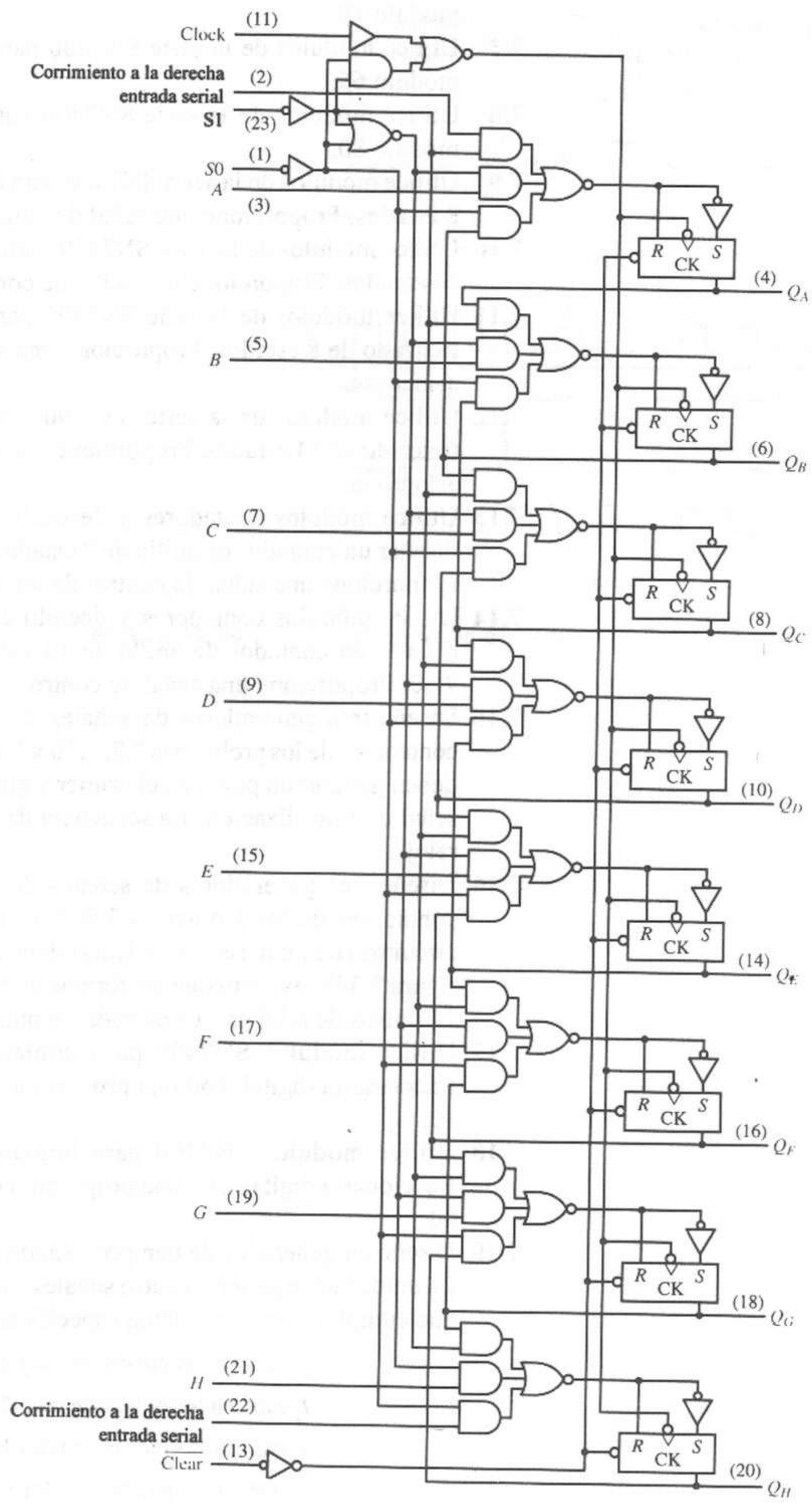


Figura P7.2 El registro de corrimiento bidireccional de 8 bits SN74198.

7.5 Utilice módulos de la serie SN7400 para diseñar un contador síncrono módulo 15.

7.6 Utilice módulos de la serie SN7400 para diseñar un contador asíncrono módulo 15

7.7 Utilice módulos de la serie SN7400 para diseñar un contador asíncrono módulo 65.

7.8 Utilice módulos de la serie SN7400 para diseñar un contador síncrono módulo 80.

7.9 Utilice módulos de la serie SN7400 para diseñar un contador de anillo con 8 estados. Proporcione una señal de control de inicialización asíncrona.

7.10 Utilice módulos de la serie SN7400 para diseñar un contador de anillo de 14 estados. Proporcione una señal de control de inicialización asíncrona.

7.11 Utilice módulos de la serie SN7400 para diseñar un contador de anillo trenzado de 8 estados. Proporcione una señal de control de inicialización asíncrona.

7.12 Utilice módulos de la serie SN7400 para diseñar un contador de anillo trenzado de 14 estados. Proporcione una señal de control de inicialización asíncrona.

7.13 Utilice módulos contadores y decodificadores de la serie SN7400 para diseñar un contador de anillo de 8 estados equivalente al de la figura 7.30. Proporcione una señal de control de inicialización asíncrona.

7.14 Utilice módulos contadores y decodificadores de la serie SN7400 para diseñar un contador de anillo de 14 estados equivalente al de la figura 7.30. Proporcione una señal de control de inicialización asíncrona.

7.15 Diseñe tres generadores de señales de tiempo equivalentes usando los contadores de los problemas 7.8, 7.10 y 7.12. Las salidas de los tres circuitos deben generar un pulso en el primer y quinto pulsos de reloj después de la señal de inicialización. La secuencia debe repetirse cada ocho pulsos de reloj.

7.16 Diseñe tres generadores de señales de tiempo equivalentes usando los contadores de los problemas 7.9, 7.11 y 7.13. Las salidas de los nueve circuitos (tres por cada problema) deben concordar con el ejemplo de la figura 7.34b, excepto que las formas de onda de tiempos se repetirán cada 14 pulsos de reloj en vez de cada 16 pulsos, como en la figura 7.34.

7.17 Utilice módulos SN7400 para implantar un multiplicador de razón fraccionaria digital, con una proporción entre la salida y la entrada de 5/10.

7.18 Utilice módulos SN7400 para implantar un multiplicador de razón fraccionaria digital, con una proporción entre la salida y la entrada de 11/80.

7.19 Diseñe un generador de tiempos usando la estructura de la figura P7.19a. La unidad debe generar cuatro señales, como se indica en la figura P7.19b, que cumplan con la siguiente especificación:

f_1 baja en los pulsos de reloj 2, 9, 17, 38 y 60

f_2 sube en los pulsos de reloj 2, 8, 15, 35 y 56

f_3 baja en los pulsos de reloj 1, 8, 16, 37 y 63

f_4 sube en los pulsos de reloj 3, 27, 39, 41 y 63

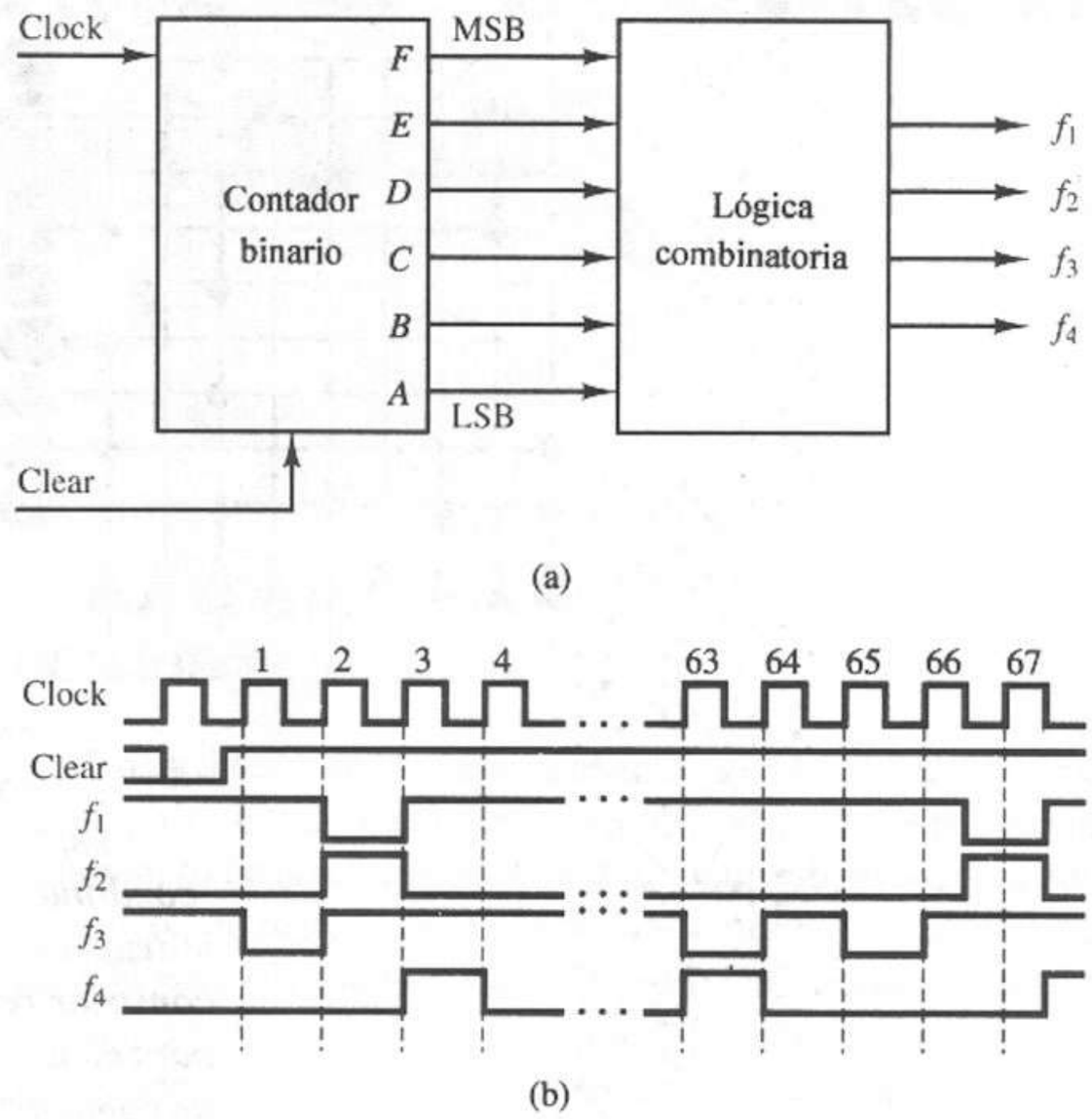
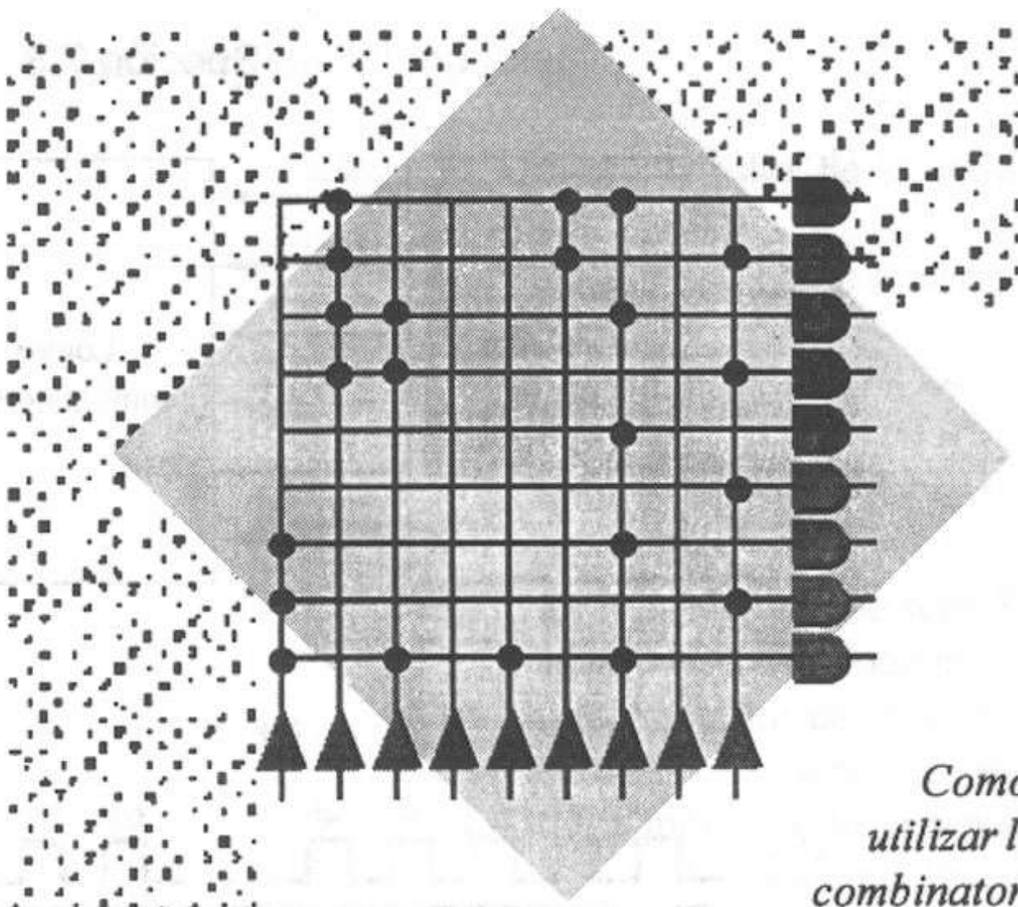


Figura P7.19 **(a)** Diagrama de bloques. **(b)** Diagrama de tiempos.

8



Como vimos en los dos últimos capítulos, podemos utilizar los dispositivos de memoria junto con la lógica combinatoria para construir muchas funciones digitales interesantes. Al pasar de funciones sencillas a otras más complejas, utilizamos una mezcla de ingenio y experiencia para combinar módulos pequeños en otros mayores. Por ejemplo, utilizamos los latches para construir flip-flops, los flip-flops para construir registros de corrimiento, y los registros de corrimiento para construir contadores de anillo trenzado. También, construimos un contador binario a partir de flip-flops JK, aprovechando nuestro conocimiento del funcionamiento del flip-flop JK en su modo de alternancia.

Suponga que un diseñador digital recibe una tabla de estados arbitraria y se le pide construir un diagrama esquemático para una implantación en hardware. Su ingenio y el conocimiento del flip-flop JK no resolverán tan fácilmente este problema. Llamamos a esto el problema general de síntesis de un circuito secuencial síncrono. En este capítulo examinaremos los métodos y herramientas para resolver tales problemas.

Análisis y síntesis de circuitos secuenciales síncronos

8.1 Modelos de circuitos secuenciales síncronos

Antes de pasar al análisis y diseño de los circuitos secuenciales, es recomendable repasar el modelo básico de circuito secuencial y cierta terminología relacionada con éste. El funcionamiento de un circuito secuencial síncrono, modelado como en la figura 8.1, es controlado por una señal de pulso de sincronización llamada *reloj*, que se aplica a la parte de memoria del circuito. Un circuito controlado por reloj es un *circuito secuencial síncrono*. Si no existe una señal de reloj, tenemos un *circuito secuencial asíncrono*. Todos los circuitos secuenciales entran en alguna de estas dos categorías. En este capítulo sólo consideraremos los circuitos síncronos; analizaremos los circuitos asíncronos en el capítulo 10.

La memoria en el diagrama de bloques de la figura 8.1 se realiza por lo general mediante *flip-flops* disparados por transición o por pulso. En las aplicaciones de circuitos secuenciales síncronos, el comportamiento de estos dos dispositivos es casi idéntico; sus salidas sólo se modifican en la transición positiva o negativa de una señal de reloj. Por tanto, el estado de un circuito secuencial síncrono sólo puede cambiar en una transición dada de señal de reloj. Para los ejemplos de este capítulo, supondremos que todos los *flip-flops* se disparan por transición.

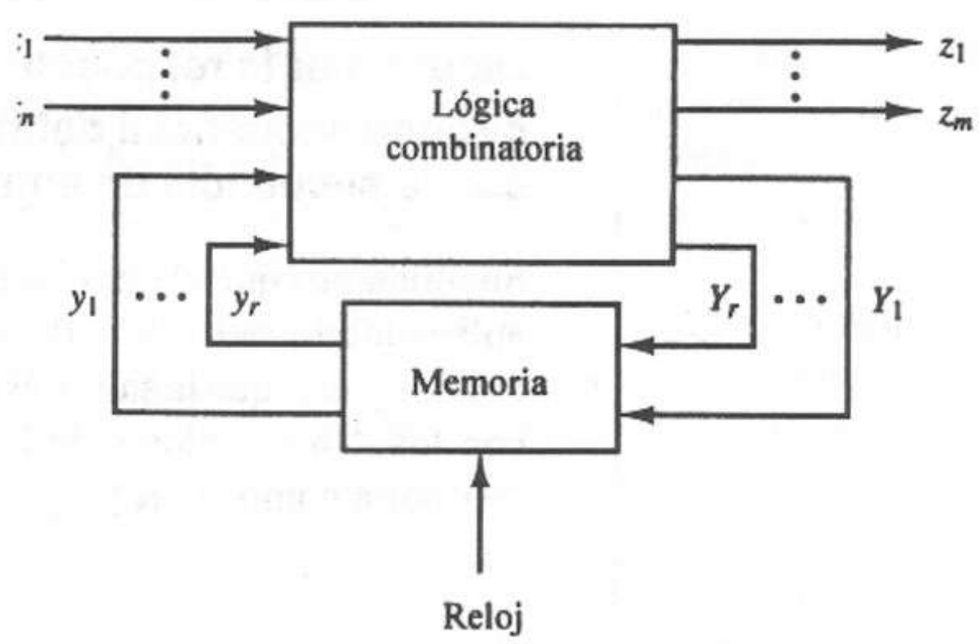


Figura 8.1 El modelo del circuito secuencial.

Recordemos del capítulo 6 que la n-ada $(x_1,...,x_n)$ de la figura 8.1 es la *entrada de circuito*, la m-ada $(z_1,...,z_m)$ es la *salida de circuito*, y las r-adas $(y_1,...,y_r)$ y $(Y_1,...,Y_r)$ son el *estado actual* y *el estado siguiente* del circuito, respectivamente. Podemos expresar las relaciones entre estas variables en forma matemática como

$$z_i = g_i(x_1, \ldots, x_n, y_1, \ldots, y_r), \; i = 1, \ldots, r \tag{8.1}$$

$$Y_i = h_i(x_1, \ldots, x_n, y_1, \ldots, y_r), \; i = 1, \ldots, r \tag{8.2}$$

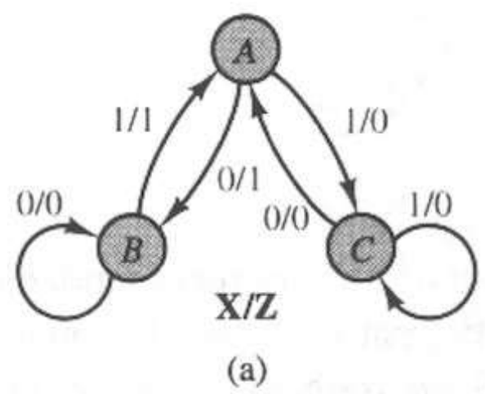
donde g_i y h_i son funciones de conmutación. Podemos escribir las ecuaciones 8.1 y 8.2 en notación vectorial como

$$\mathbf{z} = \mathbf{g}(\mathbf{x}, \mathbf{y}) \tag{8.3}$$

$$\mathbf{Y} = \mathbf{h}(\mathbf{x}, \mathbf{y}) \tag{8.4}$$

donde

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}, \; \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \; \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix}, \; \mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_r \end{bmatrix} \tag{8.5}$$



(a)

Estado actual:	Entrada x 0	1
A	$B/1$	$C/0$
B	$B/0$	$A/1$
C	$A/0$	$C/0$

Estado siguiente/salida

(b)

Figura 8.2 Circuito secuencial síncrono: Modelo de Mealy. **(a)** Diagrama de estados. **(b)** Tabla de estados.

También podemos expresar estas relaciones en forma de tablas y diagramas de estados con dos formatos básicos, según la relación entre las señales de salida z_j y las señales de entrada x_i.

8.1.1 Modelo de Mealy

En un *modelo de Mealy* de un circuito secuencial [1], las salidas son funciones de las entradas y del estado actual. Las figuras 8.2a y b muestran el diagrama de estados y la tabla de estados, respectivamente, de un circuito secuencial con el modelo de Mealy. El modelo de Mealy es un *circuito asignado a transiciones*, pues la salida del circuito está asociada a las transiciones de estado, es decir, a los arcos del diagrama de estados. En otras palabras, las salidas del circuito son funciones del estado actual y de las entradas, como muestran las ecuaciones 8.1 y 8.3. El siguiente ejemplo muestra esta relación.

EJEMPLO 8.1

Determinar la respuesta de salida del circuito secuencial definido en la figura 8.2 con la secuencia de entrada $x = 011010$.

Supongamos que el circuito está inicialmente en el estado A. En el instante 0, se aplica una entrada $x = 0$. Podemos partir del diagrama de estados o de la tabla de estados y leer que la salida es $z = 1$ y que el siguiente estado será B. Continuamos con los demás valores de la secuencia de entrada y vemos que el circuito se comporta como sigue:

Tiempo:	0	1	2	3	4	5	
Estado actual:	A	B	A	C	A	C	A
Entrada:	0	1	1	0	1	0	
Salida:	1	1	0	0	0	0	
Estado siguiente:	B	A	C	A	C	A	

Por tanto, esta secuencia de entrada aplicada a la máquina en el estado A origina la secuencia de salida

$$z = 110000$$

y deja al circuito en el estado final A.

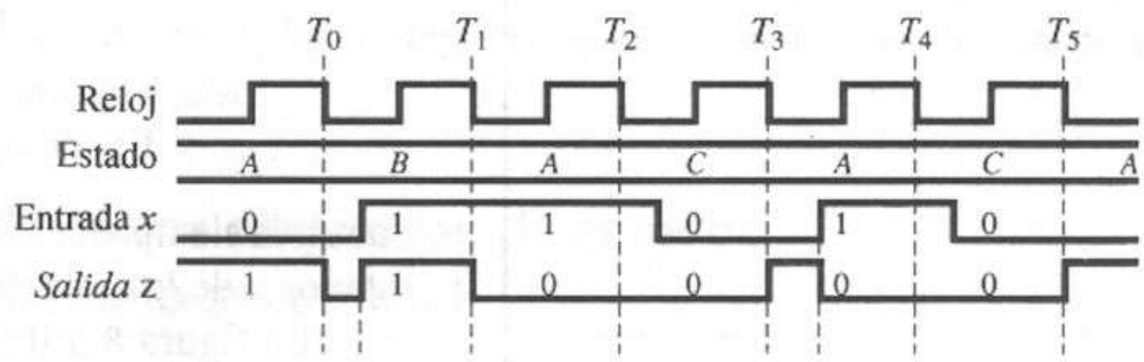
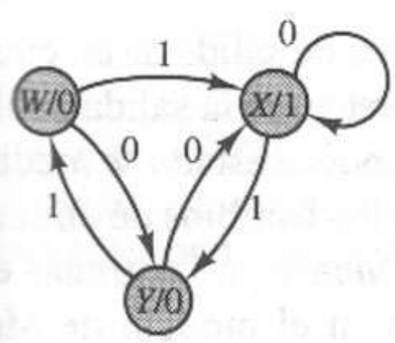


Figura 8.3 Diagrama de tiempos del modelo de Mealy.

La figura 8.3 ilustra la distribución real de tiempos para la secuencia de entrada dada. En este diagrama, suponemos que el estado cambia en la transición alta-baja del reloj. Observe que la salida z puede cambiar en cualquier momento en que cambie la entrada o el estado, pues z es función de ambos. Esto da lugar a dos cambios de salida inesperados, los cuales se muestran en el diagrama de tiempos. En el instante T_0, z baja a 0 cuando el estado cambia a B, y luego regresa a 1 cuando la entrada x cambia a 1. Ocurre un evento similar en el instante T_3. Por tanto, debemos tener cuidado de muestrear la salida de un circuito en el modelo de Mealy sólo cuando el circuito se haya estabilizado después de un cambio de entrada.



(a)

Estado actual	Entrada x 0	1	Salidas
W	Y	X	0
X	X	Y	1
Y	X	W	0

(b)

Figura 8.4 Circuito secuencial síncrono: Modelo de Moore. **(a)** Diagrama de estados. **(b)** Tabla de estados.

8.1.2 Modelo de Moore

La figura 8.4 muestra una segunda disposición para un diagrama de estados. Este formato es el *modelo de Moore* para un circuito secuencial [2] y se distingue del modelo de Mealy en que identifica las salidas sólo en términos del estado actual del dispositivo. La salida se incluye entonces dentro de los círculos que representan los estados del circuito.

La tabla de estados también tiene otro formato. Podemos eliminar la salida de las entradas de estado siguiente de la tabla de estados, pues cada estado siguiente siempre tendrá el mismo dato de salida; se muestra una nueva columna de salidas. Es importante recordar que estas salidas pertenecen al estado actual y no al siguiente.

Podemos modificar la relación funcional de salida dada en las ecuaciones 8.1 y 8.3 para los circuitos de tipo Moore como sigue:

$$z_i = g_i(y_1, \ldots, y_r), \qquad i = 1, \ldots, m \tag{8.6}$$

$$\mathbf{z} = \mathbf{g}(\mathbf{y}) \tag{8.7}$$

Esto se debe a que las salidas sólo están determinadas por el estado actual.

EJEMPLO 8.2

Consideremos la siguiente secuencia de entrada al modelo de Moore de la figura 8.4 con el estado inicial *W*:

Tiempo:	0	1	2	3	4	5	
Estado actual:	*W*	*Y*	*W*	*X*	*X*	*Y*	*X*
Entrada:	0	1	1	0	1	0	
Salida:	0	0	0	1	1	0	
Estado siguiente:	*Y*	*W*	*X*	*X*	*Y*	*X*	

La salida siempre se identifica mediante el estado actual, ya sea del diagrama de estados o de la tabla de estados.

La figura 8.5 ilustra la distribución real de tiempos para la secuencia de entrada dada. Como en el ejemplo anterior, todos los cambios de estado ocurren en la transición alta-baja del reloj.

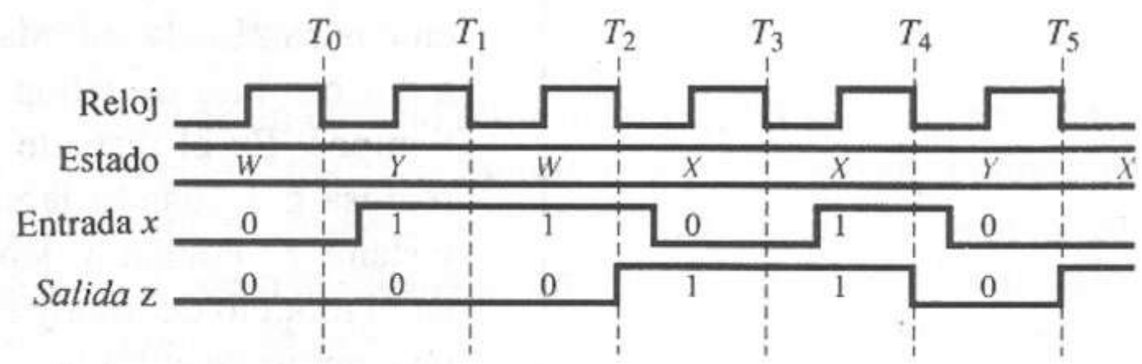


Figura 8.5 Diagrama de tiempos del modelo de Moore.

En la figura 8.5, observemos que todos los cambios de salida de un circuito en el modelo de Moore se sincronizan con el reloj, puesto que la salida *z* sólo es función del estado, y, por tanto, sólo puede cambiar cuando el estado se modifica. En consecuencia, la salida permanece estable durante los cambios de entrada, a diferencia de las salidas del circuito con modelo de Mealy ya ilustradas en la figura 8.3. Por esta razón, las salidas de un circuito en el modelo de Moore exhiben por lo general un mejor comportamiento que las de un circuito en el modelo de Mealy; es decir, los cambios de entrada no originan glitches no deseados en las salidas.

La principal ventaja del uso de un modelo de Mealy en vez de un modelo de Moore para el diseño de un circuito secuencial es que, como las salidas de un modelo de Mealy son funciones de las entradas y del estado, el diseñador tiene mayor flexibilidad en el diseño de las funciones de salida y de transición de estado, por lo que serán necesarios menos estados que en un circuito en el modelo de Moore equivalente, donde las salidas sólo son funciones de las variables de estado.

En las siguientes secciones presentaremos ejemplos de circuitos en los modelos de Mealy y de Moore.

8.2 Análisis de un circuito secuencial

Antes de abocarnos al problema de la síntesis, invertiremos el problema. Es decir, supondremos que tenemos un diagrama esquemático de un circuito secuencial síncrono y que nos piden describir su funcionamiento.

El *análisis* es el proceso de determinar la respuesta de salida de un circuito o modelo de circuito dado a una secuencia de entrada dada. Para esto, lo más conveniente es determinar primero el modelo; por ejemplo, la tabla de estados o el diagrama de estados, del circuito dado. Primero ilustraremos este proceso mediante una serie de ejemplos y después lo resumiremos en forma de un procedimiento paso por paso.

8.2.1 Análisis de los diagramas de estados de un circuito secuencial

Comenzamos con el análisis del funcionamiento de un circuito secuencial síncrono descrito por un diagrama de estados.

EJEMPLO 8.3

Consideremos el diagrama de estados dado en la figura 8.6. Supongamos que el diagrama modela un circuito secuencial síncrono que tiene como elementos de memoria *flip-flops* disparados por transición negativa.

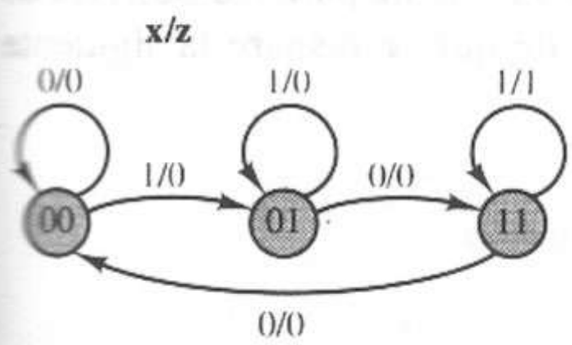


Figura 8.6 Modelo de diagrama de estados de un circuito secuencial síncrono.

Construyamos un diagrama de tiempos que ilustre el comportamiento del circuito en respuesta a la secuencia de entrada 001110110, suponiendo que el circuito inicia en el estado 00. Explicaremos el diagrama de tiempos que aparece en la figura 8.7.

En primer lugar, se necesita una señal de reloj para sincronizar las señales de entrada y los cambios de estado. Como hemos supuesto que los elementos de memoria son *flip-flops* disparados por transición negativa, las variables de estado y_1 y y_2 sólo cambian de valor en una transición $1 \rightarrow 0$ de la señal de reloj. Los valores de las variables de estado después de una transición de reloj quedan determinados por el valor de x y los valores de las variables de estado y_1 y y_2 en el instante en que ocurre la transición del reloj. En cambio, la salida z no está sincronizada con el reloj; sólo es una función combinatoria de la entrada x y de las variables de estado y_1 y y_2; por tanto, puede cambiar cuando cualquiera de estas tres señales cambie.

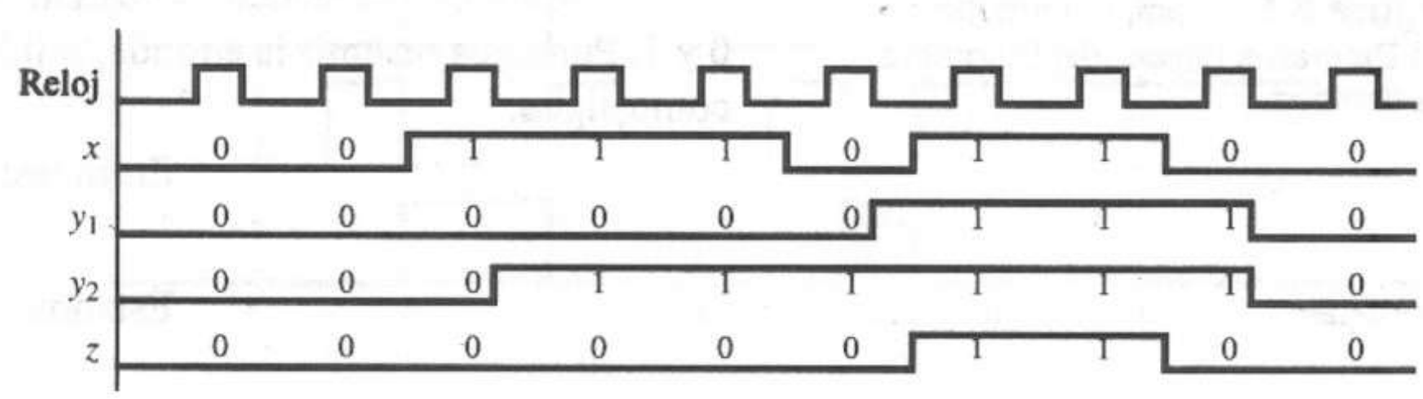


Figura 8.7 Diagrama de tiempos para un circuito secuencial síncrono.

El ejemplo anterior ilustra el siguiente punto importante; podemos construir un diagrama de tiempos que represente el comportamiento del circuito modelado mediante un diagrama de estados (o una tabla de estados) dado el tipo de circuito, el tipo de elemento de memoria, una secuencia de entrada y un estado inicial. La distribución de tiempos de las transiciones de señal de entrada no se conocen con precisión, a menos que la propia secuencia de entrada esté dada en forma de diagrama de tiempos.

8.2.2 Análisis de los diagramas lógicos de un circuito secuencial

Consideremos ahora el problema de analizar un circuito secuencial síncrono definido mediante un diagrama lógico. Para dicho circuito secuencial, necesitamos determinar la tabla de estados o diagrama de estados que define su funcionamiento. A partir de esta tabla o diagrama de estados, podremos entonces determinar la respuesta del circuito a cualquier secuencia de entrada dada.

Como ejemplo, consideremos el circuito secuencial de la figura 8.8a. Este circuito secuencial está construido mediante compuertas AND, OR y NOT, así como un *flip-flop* D. Con fines de ilustración, trazamos el circuito en la forma del modelo presentado en la figura 8.1b. Suponemos que el circuito opera de manera síncrona bajo el control de un reloj externo. En este caso, como el elemento de memoria es un *flip-flop* D disparado por transición positiva, la memoria sólo cambia de estado en una transición $0 \rightarrow 1$ de un pulso de reloj.

Tal vez logremos una mejor comprensión si primero analizamos un diagrama de tiempos para el *flip-flop* D, que se muestra en la figura 8.8b. Observe que la señal de salida Q es precisamente el valor del dato de entrada D del *flip-flop* en el momento de la transición $0 \rightarrow 1$ del reloj. Elegimos el periodo del reloj, Δt, de modo que el circuito tenga el tiempo suficiente para los cambios de estado y para aplicar las nuevas entradas antes de que se dispare la siguiente transición de estado.

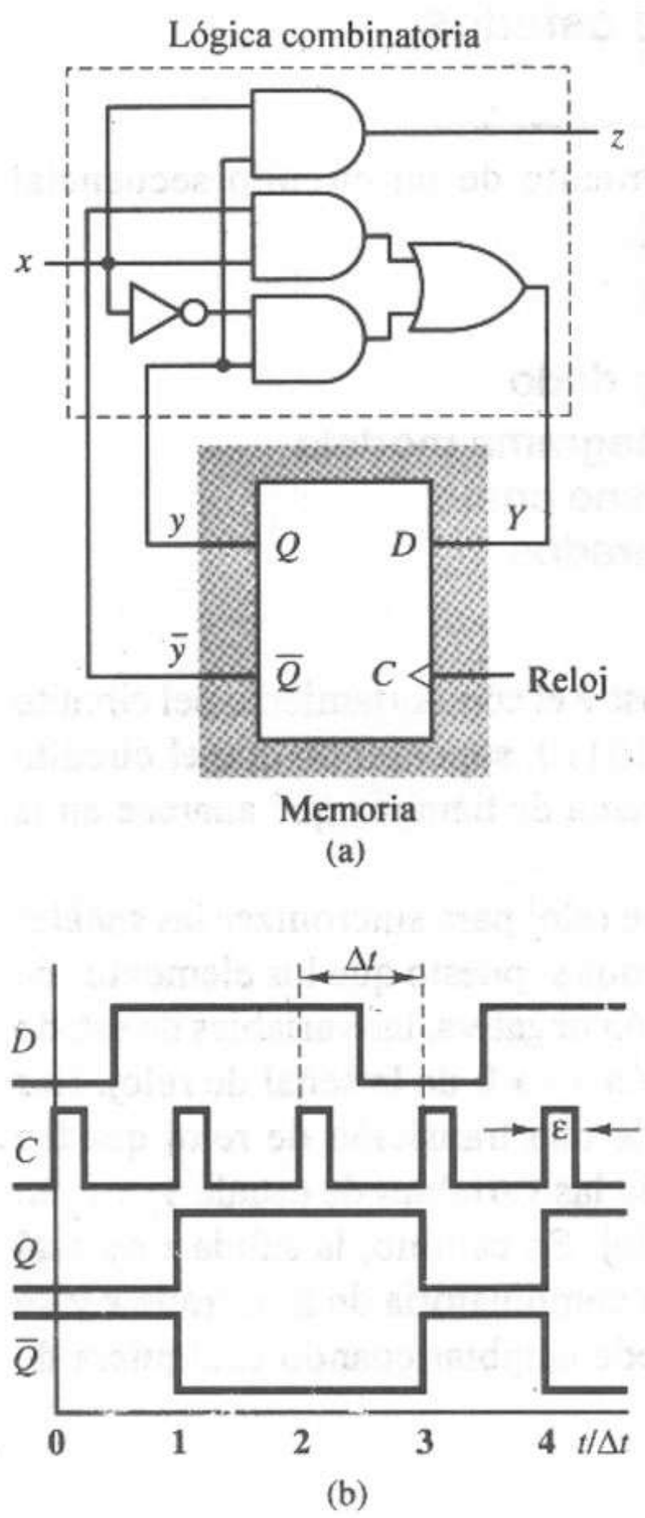


Figura 8.8 Ejemplo introductorio **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

Comportamiento de un circuito expresado como diagrama de tiempos

Ahora examinaremos el comportamiento del circuito de la figura 8.8a. En el capítulo 6 describimos el funcionamiento del *flip-flop* D, y en el capítulo 2 mostramos las características de las compuertas AND, OR y NOT. Con esta información, podemos construir un diagrama de tiempos para el circuito secuencial del ejemplo, para una secuencia de entrada dada y un estado inicial fijo.

Este circuito secuencial sólo tiene un *flip-flop* y, por tanto, sólo dos estados, 0 y 1. Podemos resumir la entrada, salida, y condiciones de estado para esta red como sigue:

Entradas:	$x = 0$
	$x = 1$
Estados:	$y = 0$
	$y = 1$
Salidas:	$z = 0$
	$z = 1$

Las ecuaciones lógicas para el ejemplo, deducidas del diagrama lógico de la figura 8.8a, son las siguientes:

$$z = xy$$
$$Y = x\bar{y} + \bar{x}y = x \oplus y \tag{8.8}$$

Podemos utilizar las ecuaciones (8.8) para construir el diagrama de tiempos de la figura 8.9. La secuencia de entrada es

$$x = 01101000$$

y el estado inicial es $y = 0$. Durante el intervalo de $t = 0$ a $t = 1$, la entrada es $x = 0$ y el estado actual es $y = 0$. Por tanto,

$$z = xy = 0 \cdot 0 = 0$$
$$Y = x \oplus y = 0 \oplus 0 = 0$$

Entonces, el pulso de reloj en $t = 1$ programa el estado siguiente $Y = 0$ en el *flip-flop* D. Durante el periodo de $t = 1$ a $t = 2$, el estado actual es $y = 0$ y la entrada cambia a $x = 1$:

$$z = xy = 1 \cdot 0 = 0$$
$$Y = x \oplus y = 1 \oplus 0 = 1$$

Cuando ocurre el pulso de reloj en $t = 2$, el estado del dispositivo secuencial cambia a 1, etcétera. Determinamos el resto del diagrama de tiempos de manera análoga.

El diagrama de tiempos nos permite observar que la secuencia de salida es

$$z = 00100000$$

En el diagrama de tiempos, observe que los cambios de Y que ocurren en $t = 3$ y $t = 5$ no afectan el estado del circuito. En cada uno de estos casos, el valor de Y cambió antes de la siguiente transición de reloj. Sólo el valor de Y durante la transición de reloj determina el estado siguiente de y. Por tanto, los cambios momentáneos en $t = 3$ y $t = 5$ son ignorados.

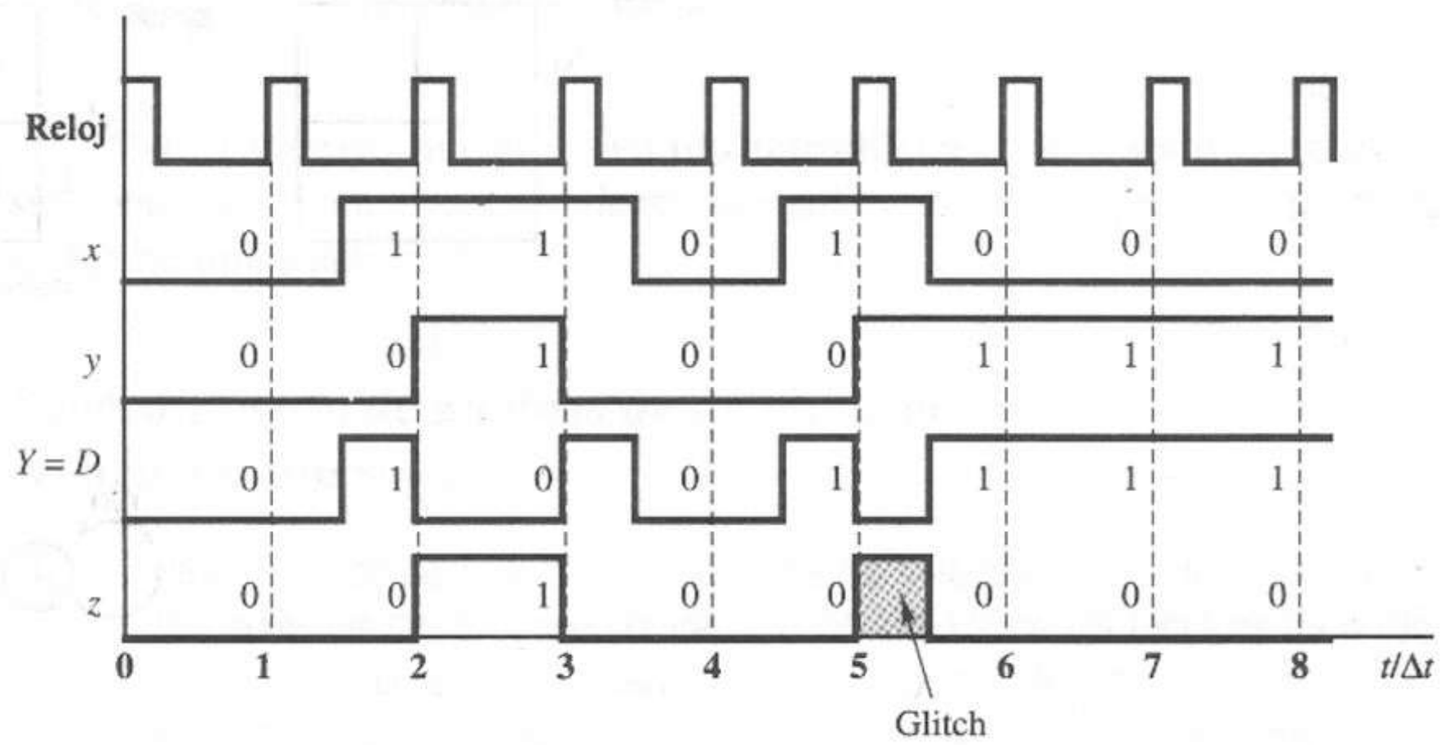


Figura 8.9 Diagrama de tiempos para el circuito de la figura 8.4(a).

El diagrama de tiempos también muestra un cambio momentáneo, o *glitch*, en la salida, en el instante $t = 5$. Como $z = xy$, el cambio en la variable de estado y en el instante $t = 5$ hace que $z = 1$ en ese instante, y después el cambio posterior en la entrada x hace que z regrese a su valor deseado, 0. Observe que el ancho de este pulso en z depende del momento en que cambia la entrada x.

Deducción del diagrama y la tabla de estados

Podemos definir completamente el funcionamiento del circuito secuencial de la figura 8.8a mediante una tabla de estados que enumere todas las condiciones operativas posibles. Adoptaremos la notación

$$y^k \quad \text{para representar} \quad y(k\,\Delta t) = y^k$$

donde k es un entero y Δt es el periodo entre los pulsos del reloj. La figura 8.10a muestra la tabla de estados en blanco. Para llenar la esquina superior izquierda, debemos suponer un estado inicial $y^k = 0$ y la entrada $x^k = 0$. Si seguimos estas señales a través del circuito de la figura 8.8a, vemos que el estado siguiente es $Y^k = y^{k+1} = 0$ y que la salida es $z^k = 0$. Por tanto, la entrada en el bloque superior izquierdo es $y^{k+1}/z^k = 0/0$.

Las condiciones iniciales para el bloque superior derecho son $y^k = 0$ y $x^k = 1$. Al aplicar estas señales en la figura 8.8a, obtenemos $z^k = 0$ y $Y^k = y^{k+1} = 1$. Por tanto, la entrada en este bloque es $y^{k+1}/z^k = 1/0$. Determinamos las dos entradas inferiores de manera análoga. La figura 8.10b muestra el resultado del análisis.

A veces reemplazamos los vectores de estado actual y mediante símbolos para simplificar la notación; por ejemplo, en la figura 8.10b podemos representar los estados como sigue:

$$\mathbf{y} = [y] = [0] \equiv A$$
$$\mathbf{y} = [y] = [1] \equiv B$$

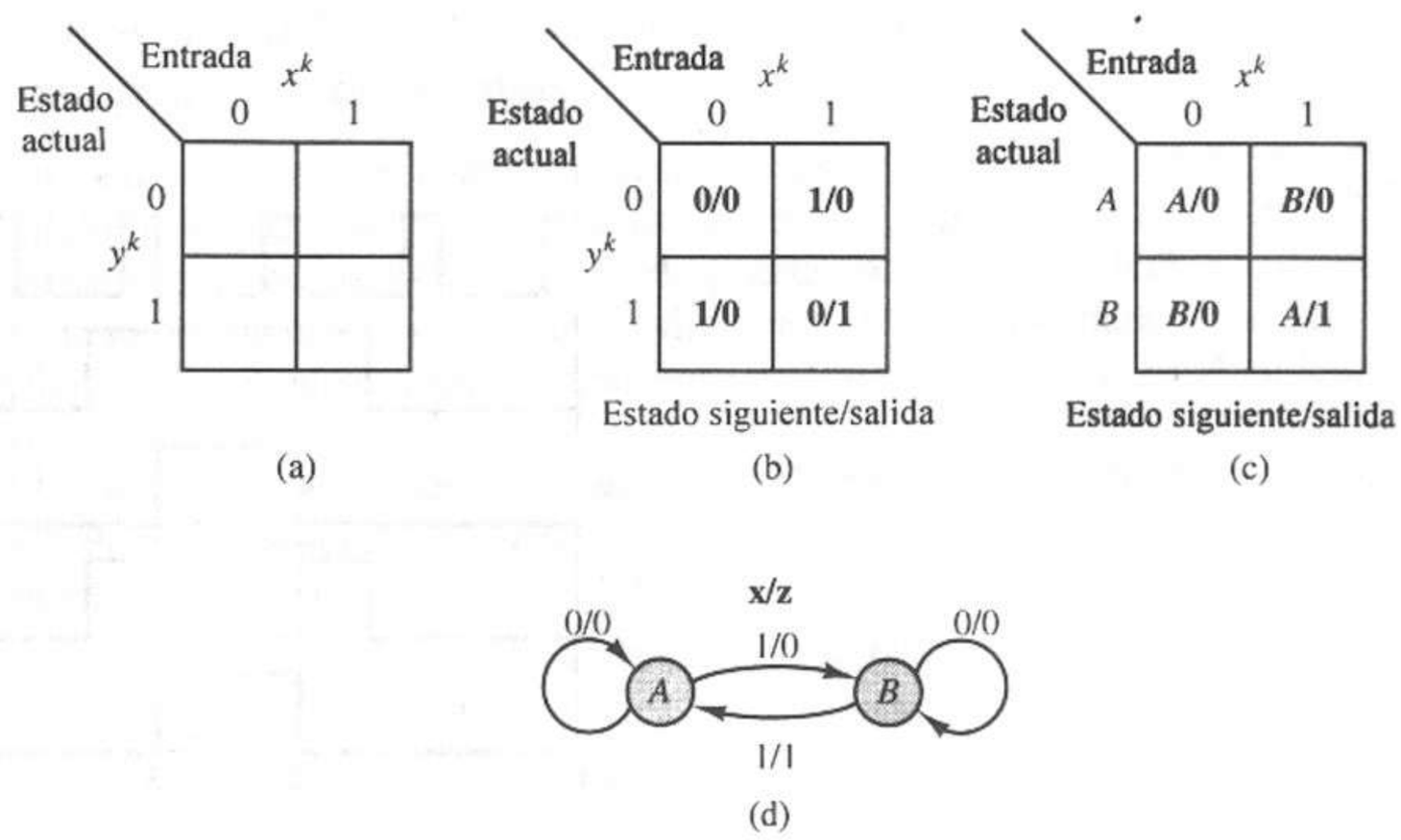


Figura 8.10 Tabla y diagrama de estados para el circuito de la figura 8.4(a). **(a)** Tabla en blanco. **(b)** Tabla de transición de estados. **(c)** Tabla de estados. **(d)** Diagrama de estados.

Esta asociación de un estado simbólico con un valor binario es una *asignación de estado*. Las figuras 8.10c y d muestran el diagrama y la tabla de estados obtenidos de la información contenida en la figura 8.10b.

Deducción de la tabla de estados a partir de los mapas de Karnaugh

Es interesante e informativo deducir la tabla de estados de la figura 8.10b directamente de las ecuaciones de circuito mediante mapas K. En la figura 8.8a observamos que

$$z = xy$$
$$Y = x \oplus y$$

Al evaluar en el instante $t = k\,\Delta t$ obtenemos

$$z^k = x^k \cdot y^k$$
$$Y^k = x^k \oplus y^k = y^{k+1}$$

Las figuras 8.11a y b muestran los mapas K para estas ecuaciones. Construimos la tabla de estados combinando los dos mapas K, como se muestra en la figura 8.11c. Esta tabla es idéntica a la que aparece en la figura 8.10b.

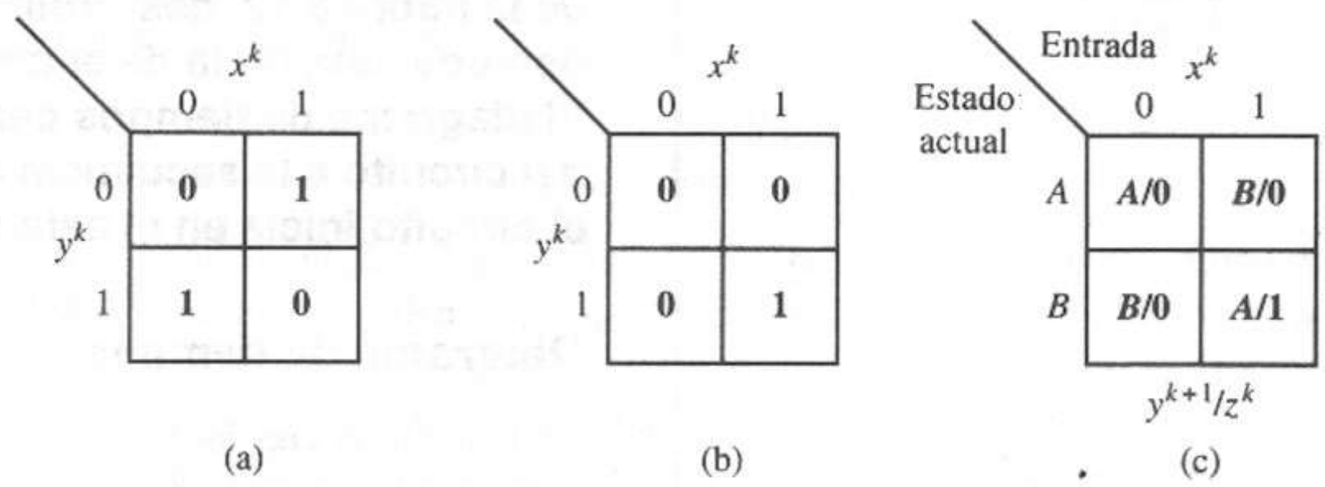


Figura 8.11 Mapas K para el circuito de la figura 8.4(a). **(a)** Mapa para $Y^k = y^{k+1}$. **(b)** Mapa para z. **(c)** Tabla de estados.

Esto concluye nuestro ejemplo introductorio de circuito secuencial síncrono. El siguiente procedimiento de análisis resume los procesos ilustrados en los ejemplos anteriores.

Procedimiento de análisis de un circuito secuencial síncrono

Paso 1. Si se da una tabla o diagrama de estados, vaya a los pasos 6 o 7, respectivamente. En caso contrario, continúe. Utilice las técnicas de análisis lógico combinatorio para determinar las ecuaciones de entrada de los *flip-flops* y las ecuaciones de salida del circuito. Vaya al paso 7 si sólo necesita un diagráma de tiempos.

Paso 2. Construya los mapas K para todas las ecuaciones lógicas del paso 1.

Paso 3. Combine los mapas K para todas las ecuaciones de entrada de *flip-flop* en un único mapa.

Paso 4. Utilice las ecuaciones características de los *flip-flops* para construir un mapa de estados siguientes.

Paso 5. Combine el mapa de estados siguientes y los mapas de salida en un único mapa. Con este paso se obtiene una tabla binaria de estados.

Paso 6. Construya un diagrama binario de estados a partir de la tabla binaria de estados si lo desea. En caso contrario, vaya al paso 7.

Paso 7. Trace un diagrama de tiempos que muestre el reloj, la secuencia de entrada dada y el estado inicial.

Paso 8. En el diagrama de tiempos, deduzca las formas de onda para las entradas y estados de *flip-flop*, para todos los valores restantes de entrada de circuito.

Paso 9. En el diagrama de tiempos, deduzca las formas de onda de las salidas del circuito.

Ilustraremos este procedimiento en los siguientes ejemplos. Comenzaremos con el problema de análisis de un circuito secuencial con reloj que contiene un *flip-flop* T.

EJEMPLO 8.4

Explorar el comportamiento del circuito de la figura 8.12, desarrollando un diagrama de tiempos, una tabla de estados y un diagrama de estados. El diagrama de tiempos debe mostrar la respuesta del circuito a la secuencia de entrada $x = 01101000$, donde el circuito inicia en el estado $y = 0$.

Diagrama de tiempos

Comenzamos con la deducción de las ecuaciones de excitación y de salida a partir del diagrama lógico de la figura 8.12, aplicando el paso 1 del procedimiento de análisis, de donde podemos generar un diagrama de tiempos mediante los pasos 7 a 9 del procedimiento.

Ya hemos mostrado antes el funcionamiento del *flip-flop* T y las características de las compuertas AND, OR y NOT. Este circuito secuencial

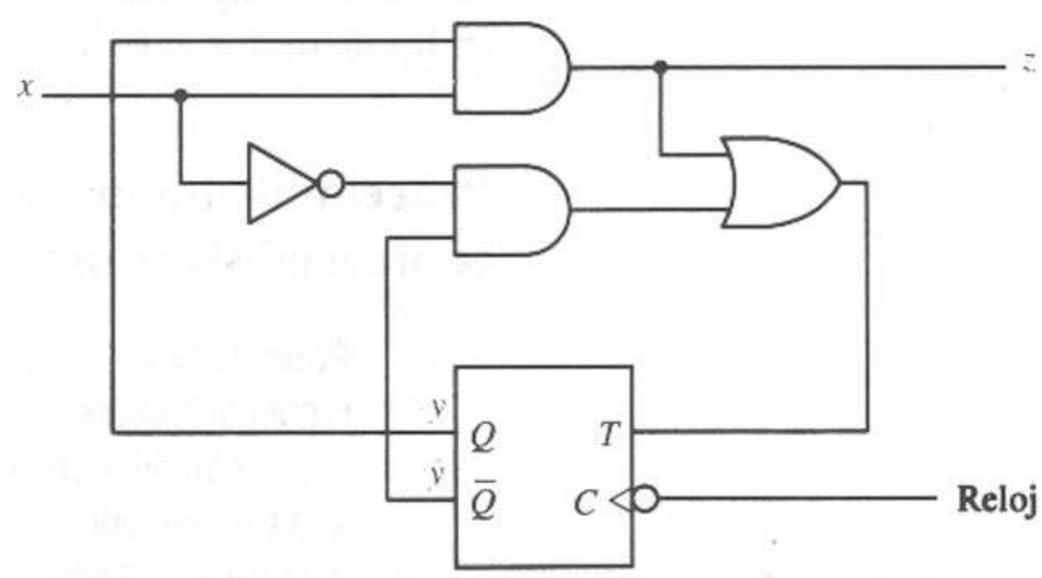


Figura 8.12 Circuito secuencial síncrono con un *flip-flop* T.

tiene sólo un *flip-flop* y, por tanto, sólo dos estados, 0 y 1. A continuación resumimos la entrada, la salida y las condiciones de estado para esta red:

Entradas:	$x = 0$
	$x = 1$
Estados:	$y = 0$
	$y = 1$
Salidas:	$z = 0$
	$z = 1$

Las ecuaciones lógicas para el ejemplo son:

$$z = xy$$
$$T = xy + \bar{x}\bar{y} = x \odot y$$

Con estas ecuaciones podemos construir el diagrama de tiempos de la figura 8.13. También utilizamos las características del *flip-flop* T con reloj descritas en el capítulo 6 para construir el diagrama de tiempos.

Un diagrama de tiempos específico para el ejemplo del circuito secuencial es válido sólo para una secuencia de entrada dada y un estado inicial fijo. La secuencia de entrada es

$$x = 01101000$$

y el estado inicial es $y = 0$. Durante el lapso de tiempo $t/\Delta t = 0$ a $t/\Delta t = 1$, la entrada es $x = 0$ y el estado actual es $y = 0$. Por tanto, si examinamos las ecuaciones lógicas durante el pulso de reloj y al final del periodo,

$$z = xy = 0 \cdot 0 = 0$$
$$T = x \odot y = 0 \odot 0 = 1$$

La variable T controla el *flip-flop* T. Durante la transición $1 \rightarrow 0$ del pulso de reloj, la señal T permite al pulso de reloj disparar el *flip-flop* al estado 1 como se muestra para y en el intervalo de $t/\Delta t = 1$ a $t/\Delta t = 2$. Determinamos todo el diagrama de tiempos de una forma similar. Observe que todas las transiciones de entrada *no* están sincronizadas con el pulso de reloj. El funcionamiento del *flip-flop* T garantiza que el valor lógico en la línea de entrada x sólo será examinado en la transición descendente del pulso de reloj. Suponga que un pulso asíncrono de entrada ocurre después del tiempo $t/\Delta t = 7$ (el pulso punteado en la figura 8.13); aunque este pulso asíncrono de entrada cambia la entrada T del *flip-flop*, es ignorado, pues ocurre mientras el reloj se encuentra en el estado lógico 0, de modo que la salida del *flip-flop* no varía.

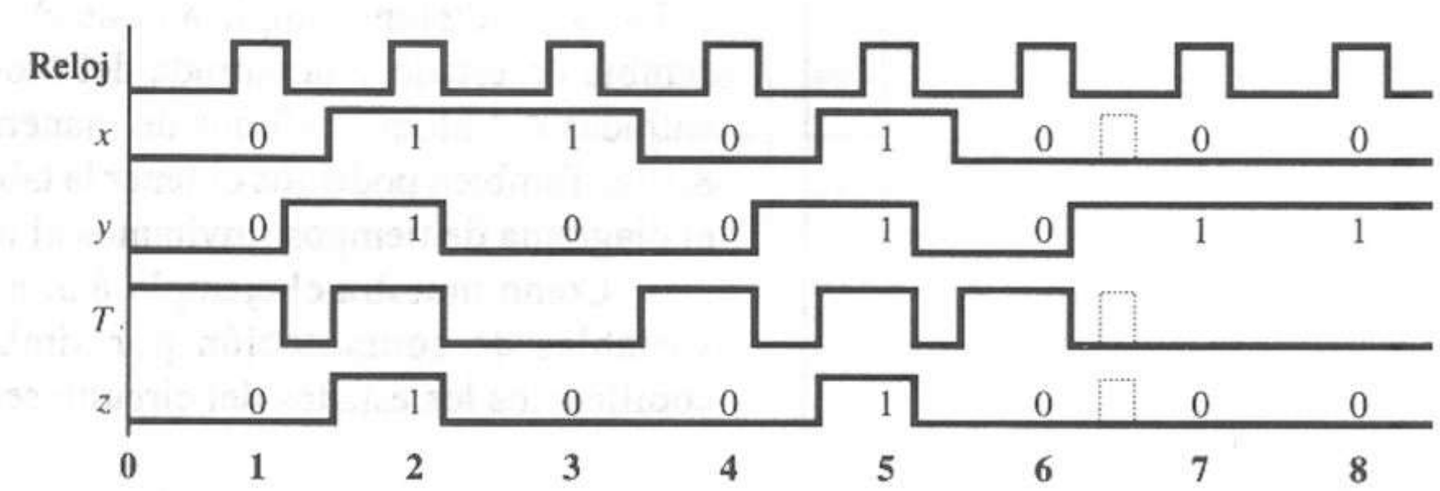


Figura 8.13 Diagrama de tiempos para el circuito secuencial síncrono con *flip-flop* T.

Diagrama de estados y deducción de la tabla de estados

Podemos deducir la tabla de estados y el diagrama de estados del circuito a partir de las ecuaciones de excitación y de salida generadas en el paso 1 del procedimiento de análisis, utilizando el paso 6 del mismo.

El funcionamiento del circuito secuencial de la figura 8.12 está definido completamente por su tabla de estados, la cual contiene todas las condiciones posibles de operación. La tabla de estados en blanco aparece en la figura 8.14a. Para llenar la esquina superior izquierda de la tabla de estados, suponemos que el estado actual es cero; es decir, $y^k = 0$, y la entrada $x^k = 0$. Las señales se examinan sólo durante el periodo $\in$, mientras el pulso de reloj es alto (1). Seguimos estas señales por el circuito de la figura 8.12 y vemos que $z^k = 0$ y $T^k = 1$, y así, durante el pulso de reloj, el *flip-flop* experimenta un cambio de estado por lo que la entrada del bloque en cuestión es $y^{k+1}/z^k = 1/0$. En otras palabras, cuando el circuito secuencial se encuentra en el estado $y^k = 0$ y la entrada es $x^k = 0$, el circuito cambia de estado, de modo que $y^{k+1} = 1$ y la salida obtenida es $z^k = 0$.

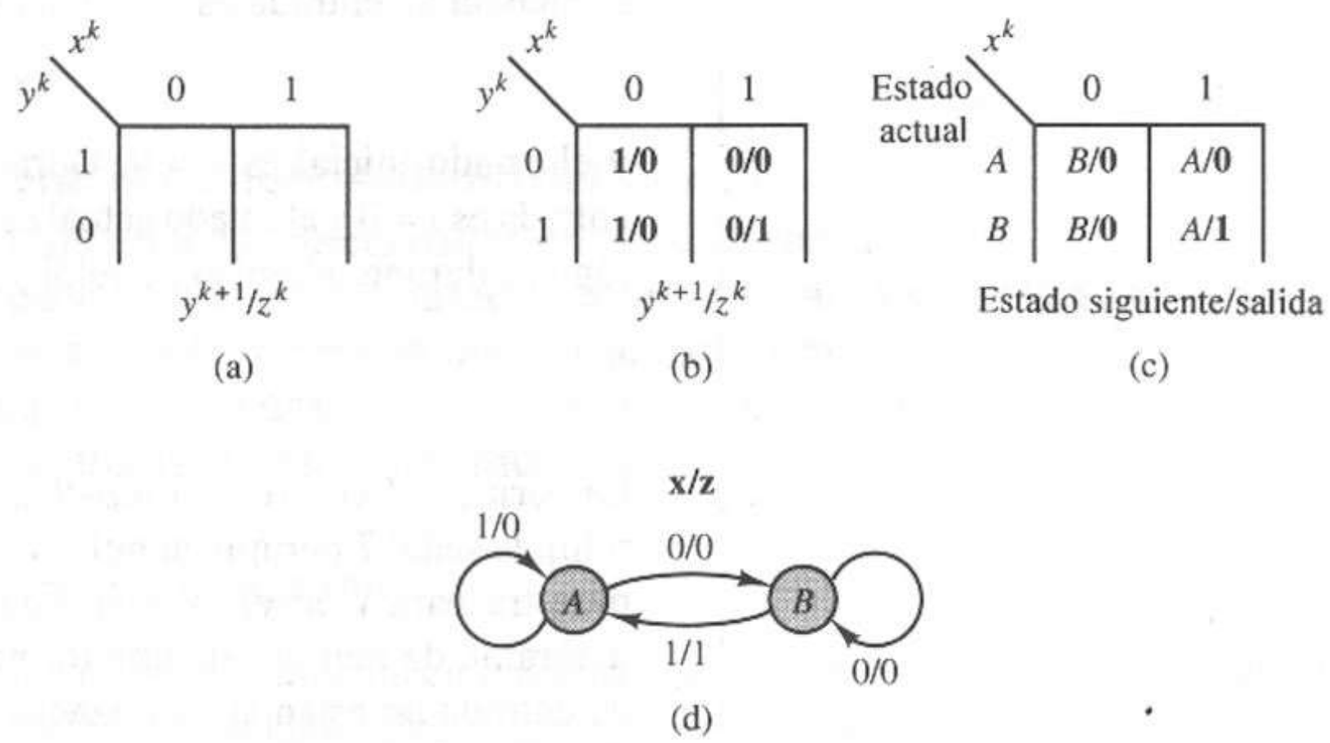


Figura 8.14 Tabla y diagrama de estados para el ejemplo del circuito secuencial con *flip-flop* T. **(a)** Tabla en blanco. **(b)** Tabla de transición. **(c)** Tabla de estados. **(d)** Diagrama de estados.

Las condiciones iniciales para el bloque superior derecho son $y^k = 0$ y $x^k = 1$. Estas condiciones implican que $z^k = 0$ y $T^k = 0$ y, por tanto, el *flip-flop* no cambia de estado y la entrada del bloque es $y^{k+1}/z^k = 0/0$. Deducimos las dos entradas del bloque inferior de manera similar, como se muestra en la figura 8.14b. También podemos obtener la tabla de estados al elegir en forma adecuada el diagrama de tiempos. Invitamos al lector a que intente este procedimiento.

Como muestra el ejemplo 8.3, a veces reemplazamos los códigos de las variables de conmutación por símbolos para simplificar la notación; si codificamos los estados del circuito secuencial como

$$0 \equiv A$$
$$1 \equiv B$$

obtenemos la tabla de estados de la figura 8.14c y el diagrama de estados de la figura 8.14d. Aquí podemos utilizar el diagrama de estados para determinar la respuesta del circuito a una secuencia de entrada:

x	=	0	1	1	0	1	0	0	0	
y	=	A	B	A	A	B	A	B	B	B
z	=	0	1	0	0	1	0	0	0	

El estado inicial es A ($y = 0$) y el final es B ($y = 1$). Observe que este comportamiento es idéntico al diagrama de tiempos de la figura 8.14, lo que era de esperar.

Deducción de la tabla de estados a partir de los mapas K

También podemos utilizar los mapas K para determinar la tabla de estados, mediante los pasos 2 a 6 del procedimiento de análisis. Las ecuaciones lógicas son

$$z^k = x^k y^k$$
$$T^k = x^k \odot y^k$$

donde y^k es el estado actual y T^k determina el estado siguiente. Los mapas K de estas ecuaciones aparecen en las figuras 8.15a y b. El mapa para T^k define la señal a la entrada T del *flip-flop* cuando hay un pulso de reloj para diversas condiciones de x y y. $T = 1$ hace que el *flip-flop* cambie de estado cuando se presenta un pulso de reloj; para $T = 0$, no hay cambio de estado.

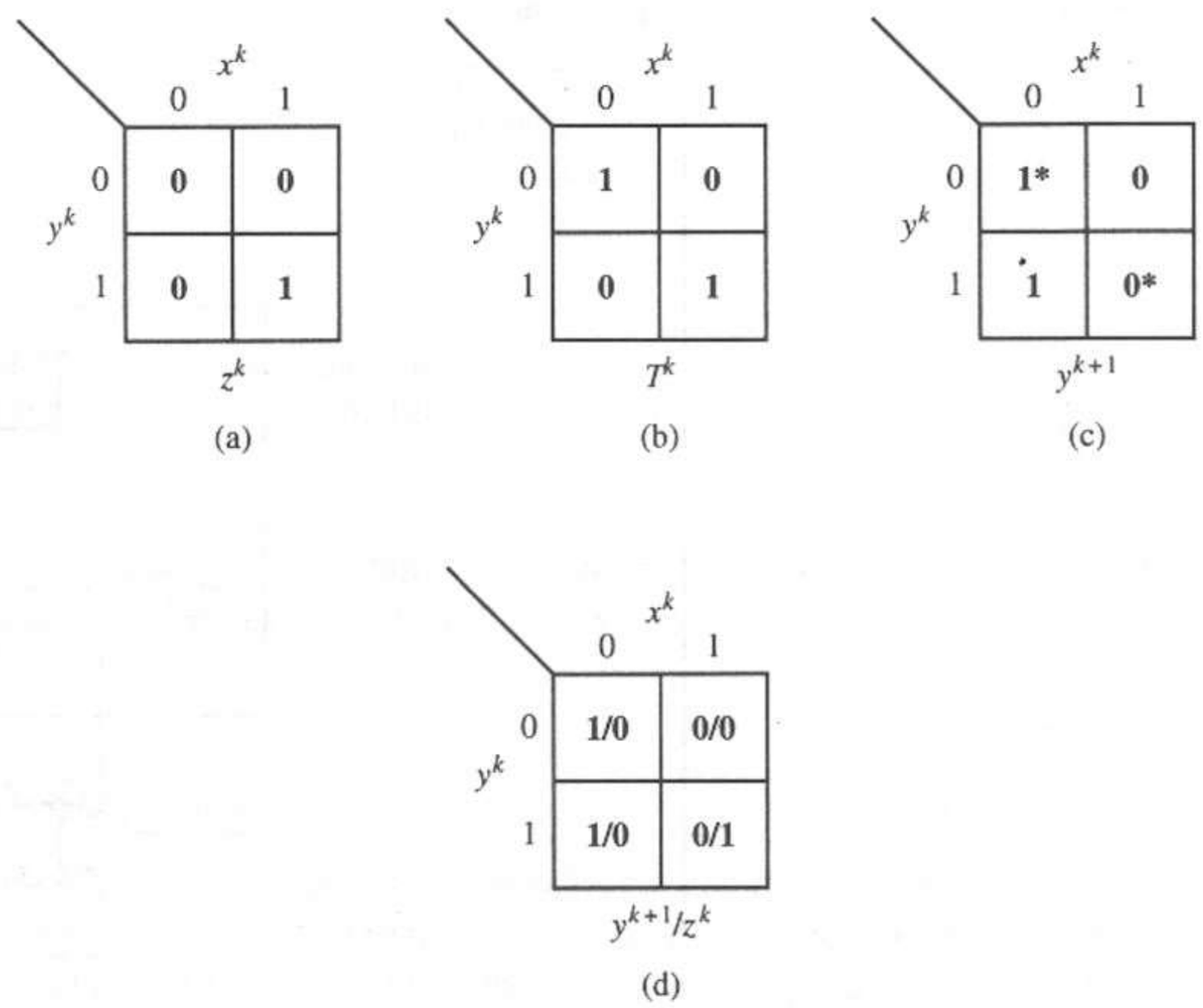


Figura 8.15 Deducción de la tabla de estados a partir de los mapas K. **(a)** Salida del mapa K. **(b)** Mapa K de excitación. **(c)** Mapa de estado siguiente. **(d)** Tabla binaria de estados.

Con base en el mapa para T^k de la figura 8.15b, debemos encontrar ahora el mapa para el estado siguiente y^{k+1}. Obtenemos este mapa en la figura 8.15c. Las variables del mapa K son la entrada x^k y el estado actual y^k. Obtenemos la entrada del estado siguiente al examinar las entradas correspondientes en el mapa para T^k. Los asteriscos indican los cambios de estado causados por los mintérminos $T = 1$.

Por último, combinamos los mapas K del estado siguiente y de la salida de las figuras 8.15a y c para obtener la tabla de estados de la figura 8.15d, que es idéntica a la de la figura 8.14b. Esto concluye el ejemplo.

EJEMPLO 8.5

Analizar el circuito de la figura 8.16.

El circuito tiene dos *flip-flops* JK disparados por transición negativa con reloj y, por tanto, cuatro estados. Con un análisis similar al anterior, obtenemos el diagrama de tiempos y la tabla de estados de la figura 8.17. La secuencia de entrada y el estado inicial son

$$x = 0011110$$
$$y_1^0 y_2^0 = 10$$

Las ecuaciones que describen el funcionamiento del circuito son

$$J_1 = xy_2, \quad J_2 = x, \quad z = xy_1y_2$$
$$K_1 = \bar{x}, \quad K_2 = \bar{x} + \bar{y}_1,$$

La figura 8.18 muestra los mapas K para estas ecuaciones. La figura 8.19a muestra una tabla donde combinamos los mapas K para J y K. Utilizamos las señales J y K para determinar los cambios de estado y obtener la tabla de la figura 8.19b.

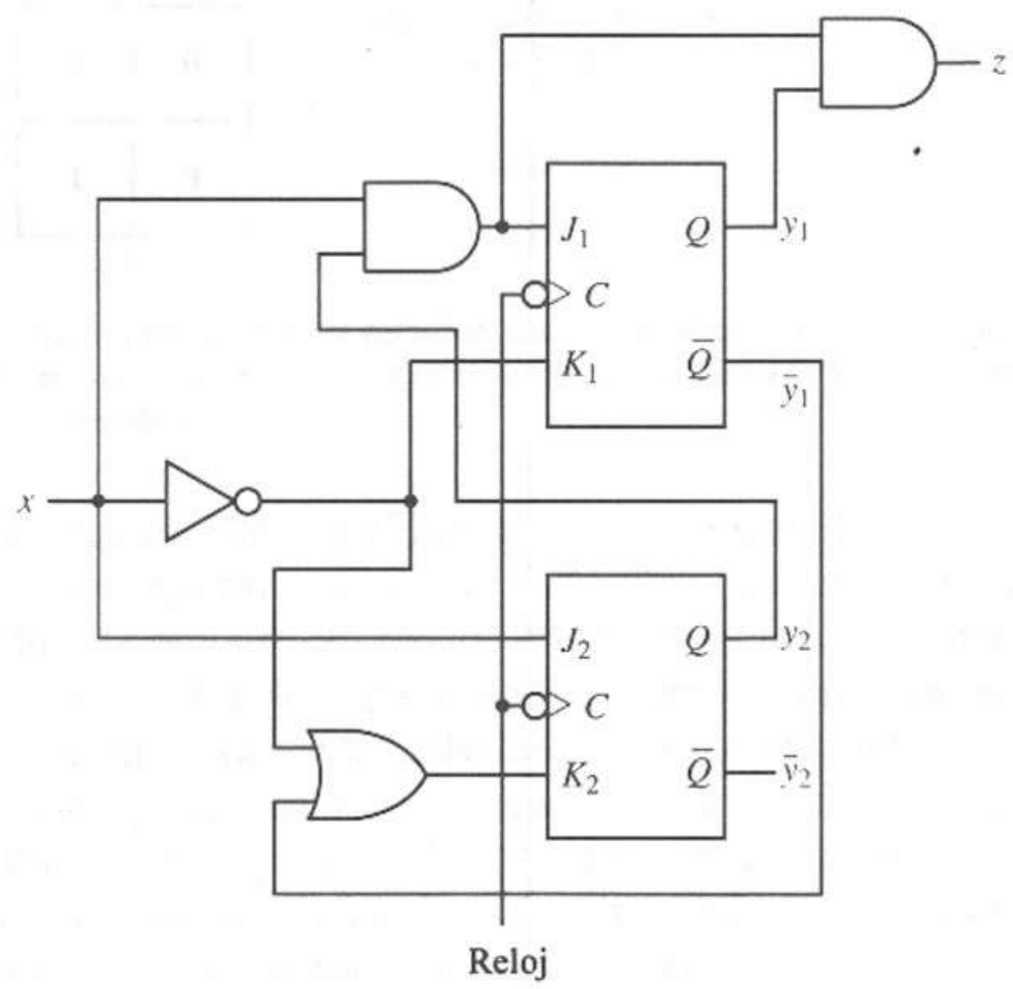
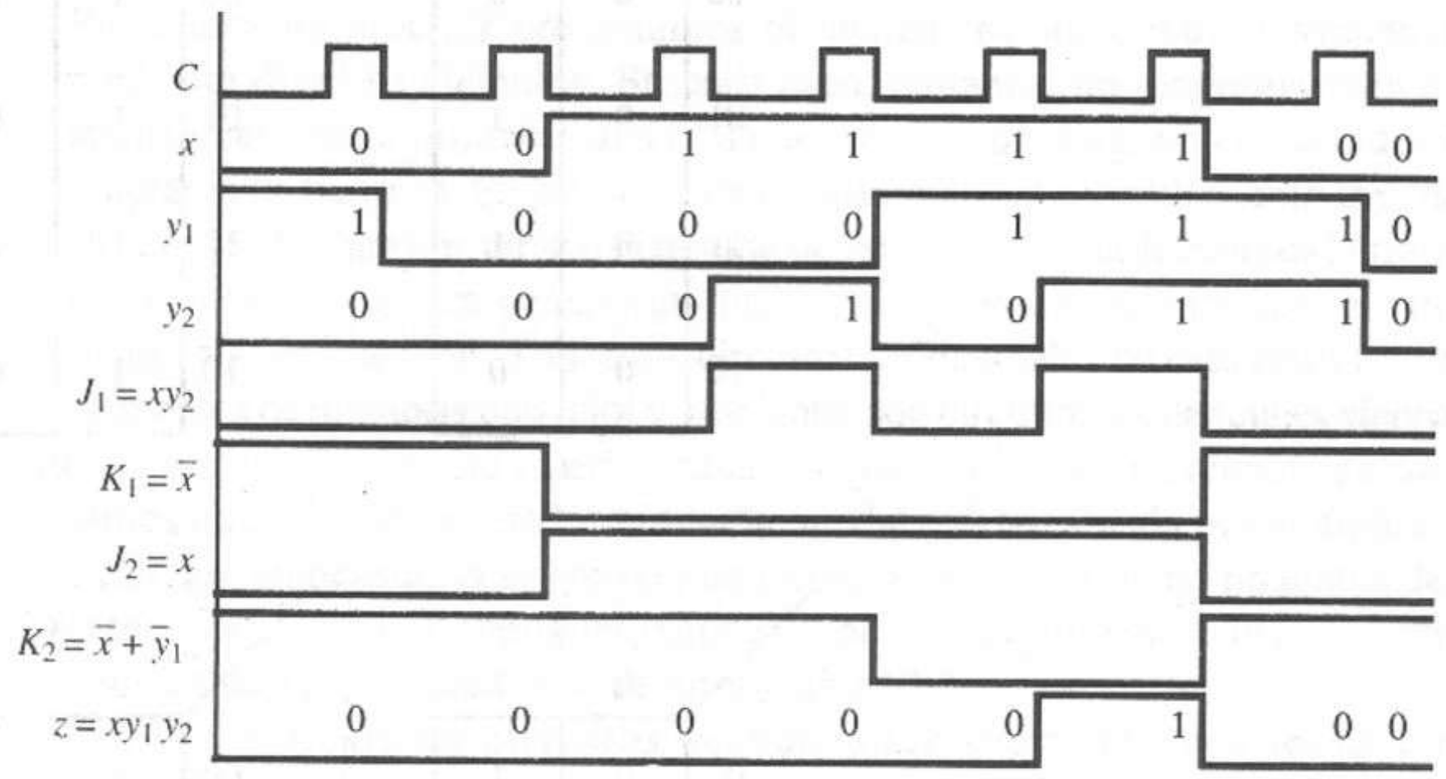


Figura 8.16 Circuito secuencial síncrono con *flip-flops* JK.



(a)

$y_1 y_2$ \ x	0	1
00	**00/0**	**01/0**
01	**00/0**	**10/0**
11	**00/0**	**11/1**
10	**00/0**	**11/0**

(b)

Figura 8.17 Diagrama de tiempos y tabla de estados para el circuito de ejemplo. **(a)** Diagrama de tiempos. **(b)** Tabla de estados.

Por último, al combinar la tabla de transición y la tabla para la salida z obtenemos la tabla binaria de estados de la figura 8.19c.

8.2.3 Resumen

En esta sección examinamos los métodos para analizar diversos tipos de circuitos secuenciales síncronos. Ahora, el lector debe poder construir el diagrama lógico para cualquier red secuencial dada (si ésta no es demasiado compleja), aplicar las técnicas de análisis de este capítulo y deducir una descripción del circuito mediante una tabla de estados o un diagrama de tiempos. En la siguiente sección invertiremos este procedimiento y determinaremos los diagramas de circuitos lógicos que realicen una tabla o diagrama de estados específico.

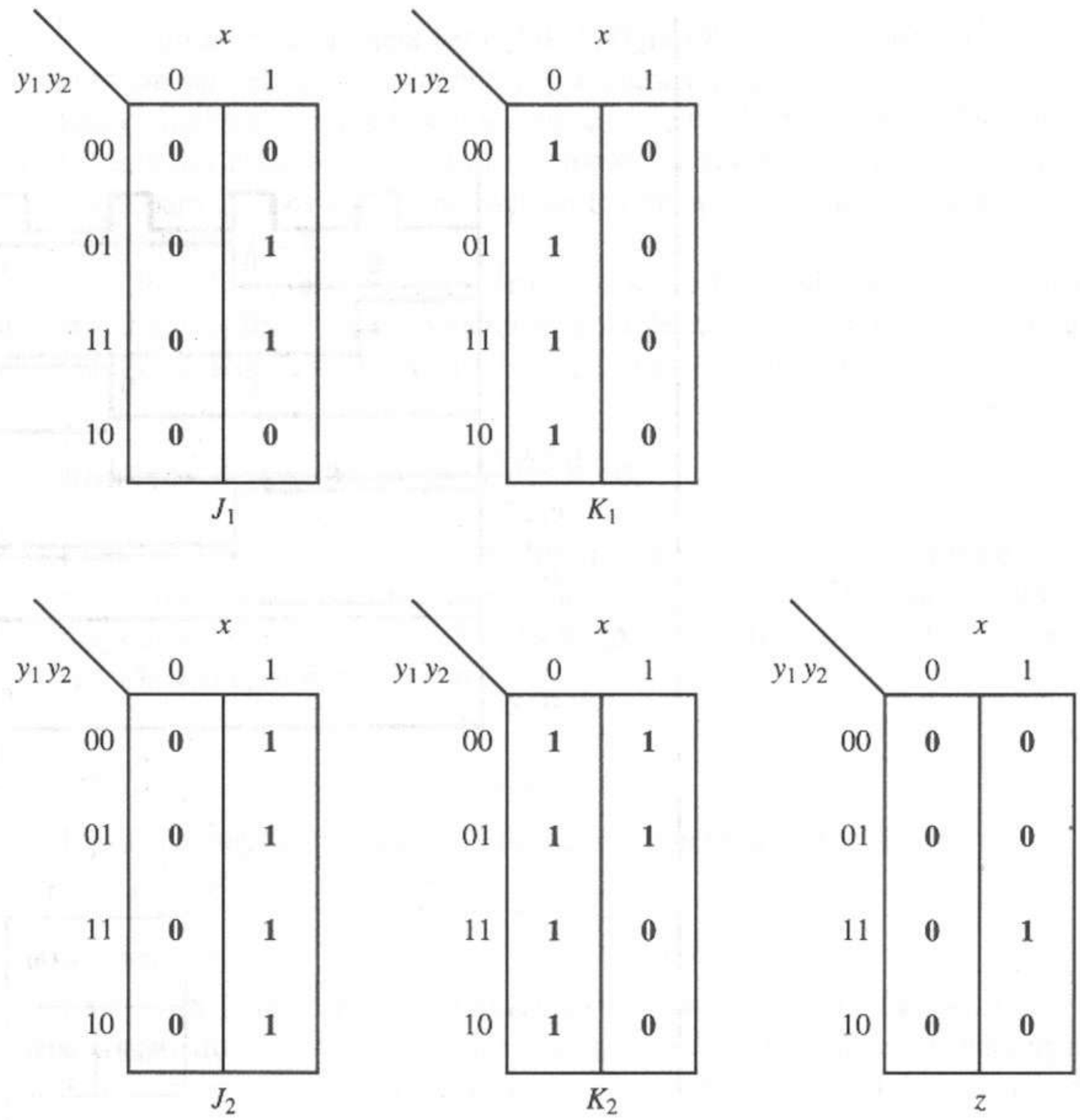


Figura 8.18 Mapas K para las ecuaciones lógicas que describen el circuito de ejemplo.

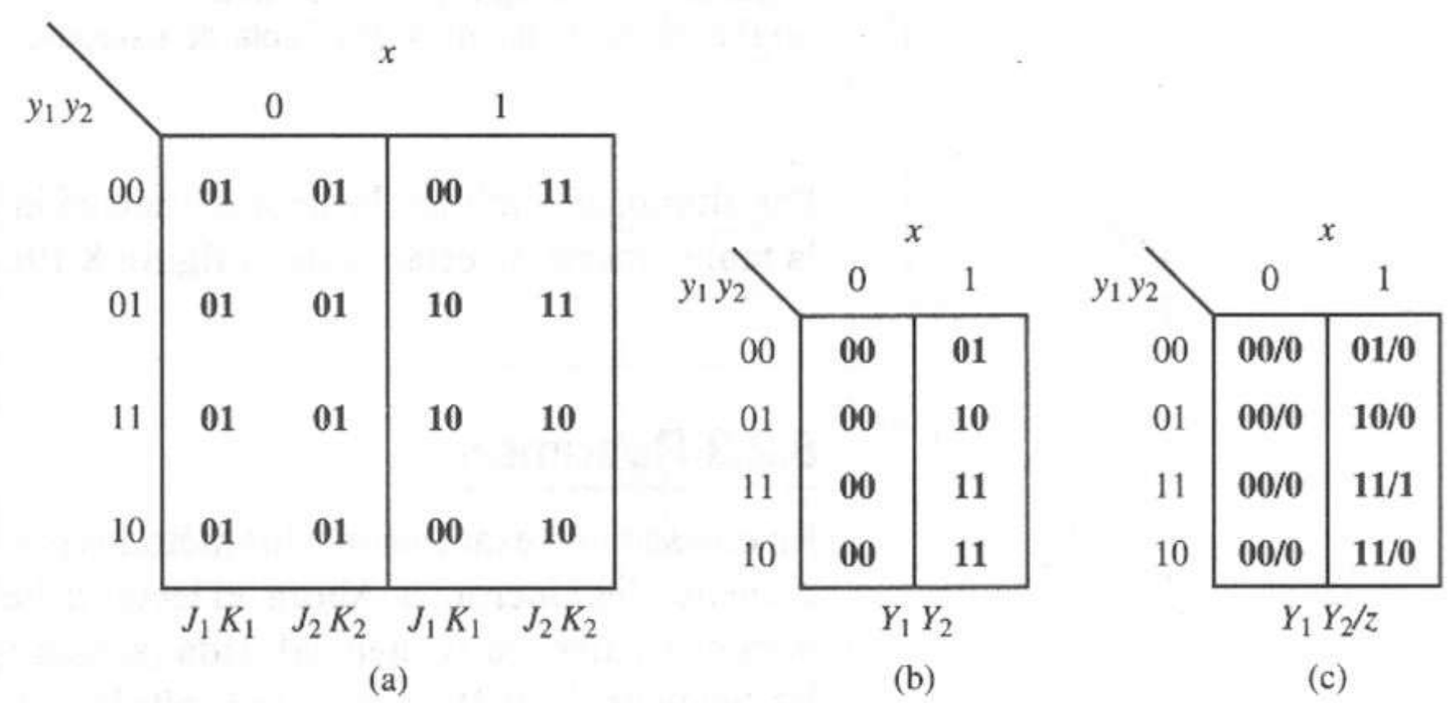


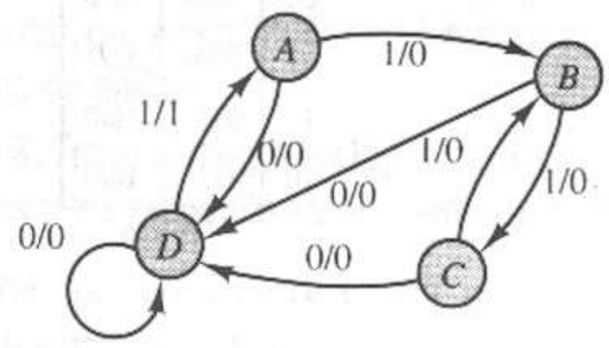
Figura 8.19 Combinación de los mapas K en la tabla de estados.

8.3 Síntesis de un circuito secuencial síncrono

En la sección anterior presentamos el análisis de los circuitos secuenciales mediante diversos ejemplos. En cada caso, teníamos un diagrama lógico y el análisis resultante producía una tabla de estados, un diagrama de estados o un diagrama de tiempos. En esta sección estudiaremos el procedimiento inverso, la síntesis [3-9]. Para una tabla o diagrama de estados dado, utilizaremos herramientas bien definidas para generar un diagrama lógico equivalente para el circuito secuencial en cuestión. Todos los circuitos secuenciales de este capítulo tienen elementos de memoria con reloj y, por tanto, son circuitos secuenciales *síncronos*.

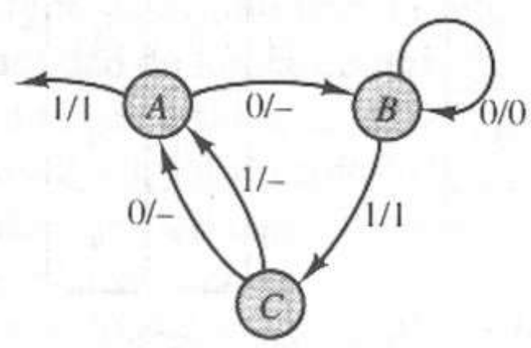
En nuestro trabajo anterior, cada vez que analizamos un circuito secuencial vimos que la tabla de estados obtenida quedaba determinada por completo. Sin embargo, en ocasiones se conecta un circuito de modo que no podemos definir la tabla de estados por completo (por ejemplo, un circuito que causa una entrada 1 en las dos terminales *S* y *R* de un *flip-flop* SR).

La síntesis de circuitos secuenciales síncronos comienza con la especificación de la tabla (o diagrama) de estados deseada. Los circuitos tales que cada pareja estado siguiente/salida está definida por completo son *circuitos con especificación completa*. Los circuitos con varios estados siguientes o salidas arbitrarios son *circuitos con especificación incompleta*. La figura 8.20 muestra un ejemplo de cada tipo de circuito. En las siguientes secciones estudiaremos



	x = 0	x = 1
A	D/0	B/0
B	D/0	C/0
C	D/0	B/0
D	D/0	A/1

(a)



	x = 0	x = 1
A	B/–	–/1
B	B/0	C/1
C	A/–	A/–

(b)

Figura 8.20 Tipos de circuitos secuenciales. **(a)** Circuito con especificación completa. **(b)** Circuito con especificación incompleta.

los circuitos con especificación completa. Los circuitos con especificación incompleta se verán en una sección posterior del capítulo.

8.3.1 Procedimiento de síntesis

Presentaremos el procedimiento para el diseño de circuitos secuenciales síncronos mediante un ejemplo sencillo.

EJEMPLO 8.6

Determinar una realización con *flip-flops* D con reloj para el circuito secuencial definido mediante la tabla de estados de la figura 8.21a.

Primero adoptaremos un esquema de codificación para los estados simbólicos. Este proceso es la *asignación de estados*. Elegimos en forma arbitraria el código de la figura 8.21b. Si reemplazamos los estados simbólicos por sus códigos equivalentes, obtenemos una tabla binaria de estados o, en otras palabras, la *tabla de transición* de la figura 8.21c. Esta tabla contiene toda la información necesaria para generar las funciones de conmutación de la parte lógica combinatoria del circuito. Después separamos la tabla de transiciones en un mapa K de salidas y los mapas K de las entradas del *flip-flop* D como en las

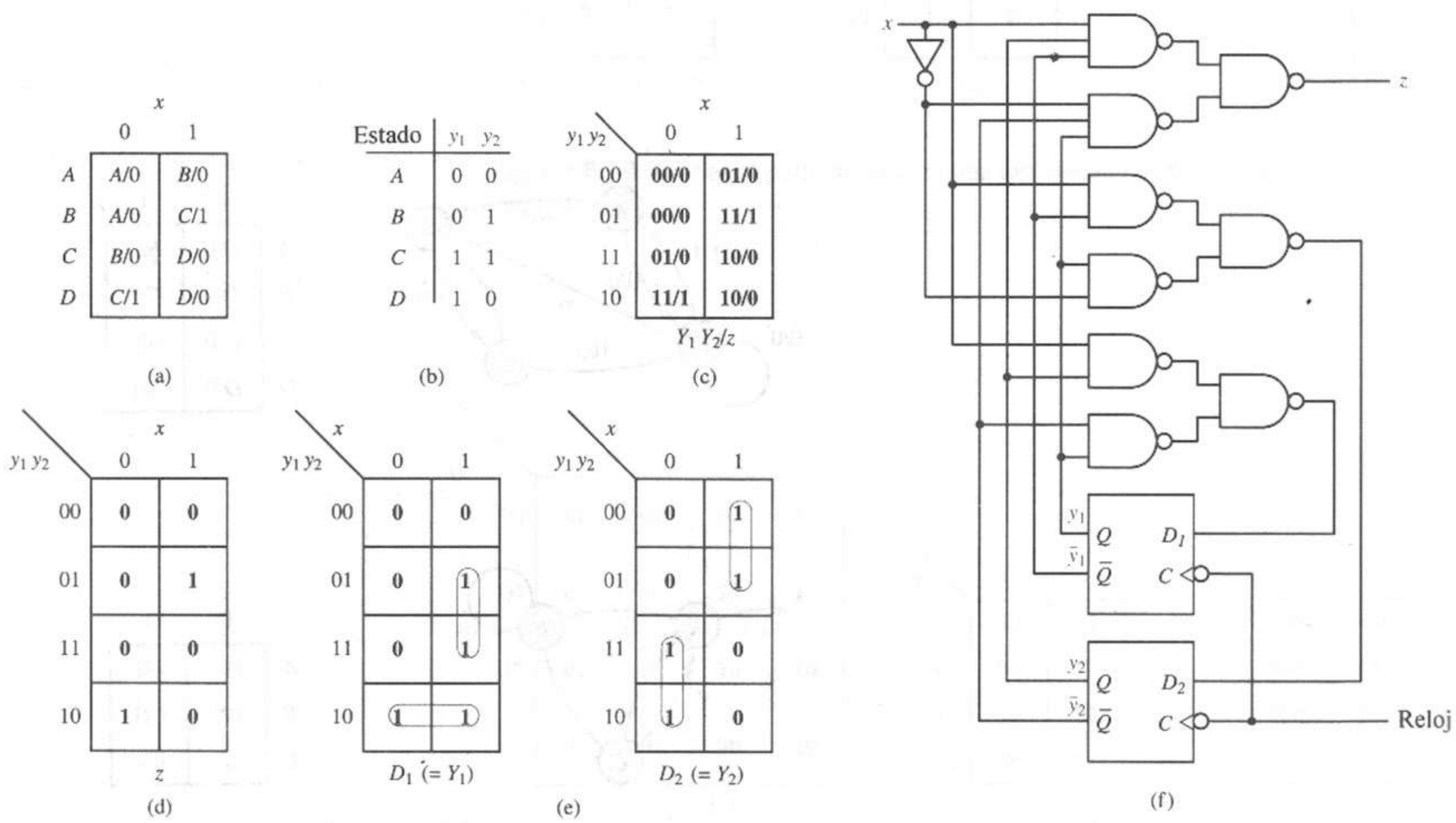


Figura 8.21 Ejemplo introductorio. **(a)** Tabla de estados. **(b)** Asignación de estados. **(c)** Tabla de transición. **(d)** Mapa K de salida. **(e)** Mapas K de excitación. **(f)** Diagrama lógico.

figuras 8.21d y e. Los mapas de entrada del *flip-flop* son los *mapas de excitación*. De los mapas de excitación y de salida, tenemos que

$$D_1 = y_1\bar{y}_2 + xy_2$$
$$D_2 = \bar{x}y_1 + x\bar{y}_1 = x \oplus y_1$$
$$z = x\bar{y}_1y_2 + \bar{x}y_1\bar{y}_2$$

La figura 8.21f muestra el diagrama lógico para el diseño completo. Realizamos la lógica combinatoria con dos niveles de compuertas NAND.

En el ejemplo anterior dejamos varias preguntas sin responder. ¿Cómo elegimos la asignación de estados? ¿Qué sucede si el diseño requiere otro tipo de *flip-flop*? ¿Cómo sabemos que la tabla de estados dada es la mejor? Muchas tablas de estados tienen estados adicionales que podemos eliminar.

Antes de intentar responder a estas preguntas, describiremos el procedimiento completo de síntesis para los circuitos lógicos secuenciales síncronos.

Procedimiento de síntesis de un circuito secuencial síncrono

Paso 1. Deduzca una tabla de estados a partir de una descripción verbal del problema.

Paso 2. Utilice las técnicas de reducción de estados para determinar la tabla de estados de un circuito equivalente con un mínimo de estados.

Paso 3. Elija una asignación de estados y genere las tablas de transición de estados y salidas.

Paso 4. Determine el dispositivo de memoria o *flip-flop* por utilizar, así como los mapas de excitación del *flip-flop*.

Paso 5. Utilice los mapas de excitación para obtener las ecuaciones lógicas de conmutación. Además, forme los mapas de salida y determine las ecuaciones lógicas de salida.

Paso 6. Trace el diagrama lógico del circuito secuencial mediante las ecuaciones lógicas y los dispositivos de memoria elegidos.

El primer paso requiere cierta intuición por parte del diseñador lógico, quien debe desarrollar su experiencia mediante un proceso de prueba y error. Utilizamos el paso 2 para minimizar el número de elementos de memoria necesarios para construir un circuito, eliminando los estados innecesarios de la tabla de estados. En el paso 3 podemos elegir una asignación de estados arbitraria o bien aplicar algún algoritmo para la selección de la asignación de estados que optimice la cantidad de lógica combinatoria necesaria para realizar un circuito. Veremos que un buen diseñador utiliza algoritmos y reglas empíricas para llevar a cabo los pasos 2 y 3. En el capítulo 9 examinaremos estos algoritmos y reglas, después de adquirir cierta experiencia con los fundamentos del diseño de un circuito secuencial síncrono.

El paso 4 requiere el análisis de las características de los tipos de *flip-flops* seleccionados, presentadas en el capítulo 6, para deducir los mapas de excitación del *flip-flop*. El capítulo 3 muestra las habilidades necesarias para el paso 5, por lo que supondremos que el lector ya está familiarizado con ellas. El paso 6 es la conclusión obvia del procedimiento de síntesis y lo incluimos para que la exposición sea más completa.

Ahora que hemos definido los problemas, examinaremos en primer lugar las técnicas para el diseño de circuitos lógicos que realizan una tabla o diagrama de estados dado. Después enfrentaremos el problema de la creación de tablas y diagramas de estados a partir de la descripción verbal de los problemas. En el capítulo 9 estudiaremos los métodos de minimización utilizados en los pasos 2 y 3.

8.3.2 Tablas de entrada de un *flip-flop*

En el ejemplo de síntesis introductorio de la figura 8.21 utilizamos un *flip-flop* D disparado por transición para realizar el circuito. En particular, utilizamos las características del *flip-flop* D para generar los mapas de excitación a partir de la tabla de transición. La tabla de transición define las transiciones de estado necesarias para cada *flip-flop* de memoria. Podemos utilizar una *tabla de entrada de flip-flop* para determinar las entradas de excitación necesarias para cada tipo de elemento de memoria *flip-flop*. Considere las tablas de entrada de la figura 8.22, las cuales representan las características de los cuatro tipos principales de *flip-flop* analizados en el capítulo 6. La notación de esta tabla es la siguiente: t es el instante en que se activa la señal de reloj; $Q(t)$ es el estado del *flip-flop* cuando se activa la señal de reloj; $Q(T+\in)$ es el estado siguiente del *flip-flop* después de que la señal de reloj ha sido activada.

Es conveniente utilizar el *flip-flop* D, ya que su estado siguiente es tan sólo su entrada actual; por tanto, obtenemos los mapas de excitación de manera directa a partir de la tabla de transición. Para obtener los mapas K de excitación de cualquier otro *flip-flop* debemos aplicar la tabla de entrada correspondiente

Transiciones de estado		Entradas necesarias
$Q(t)$	$Q(t+\varepsilon)$	$D(t)$
0	0	0
0	1	1
1	0	0
1	1	1

(a)

Transiciones de estado		Entradas necesarias	
$Q(t)$	$Q(t+\varepsilon)$	$S(t)$	$R(t)$
0	0	0	d
0	1	1	0
1	0	0	1
1	1	d	0

(b)

Transiciones de estado		Entradas necesarias
$Q(t)$	$Q(t+\varepsilon)$	$T(t)$
0	0	0
0	1	1
1	0	1
1	1	0

(c)

Transiciones de estado		Entradas necesarias	
$Q(t)$	$Q(t+\varepsilon)$	$J(t)$	$K(t)$
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

(d)

Figura 8.22 Tablas de entrada de los *flip-flops*. **(a)** *Flip-flop* D. **(b)** *Flip-flop* SR con reloj. **(c)** *Flip-flop* T con reloj. **(d)** *Flip-flop* JK con reloj.

de la figura 8.22. Las tablas de entrada del *flip-flop* se deducen de las ecuaciones características del *flip-flop* y de las tablas de excitación que se presentaron en el capítulo 6.

EJEMPLO 8.7

Implantar el circuito secuencial de la figura 8.21 mediante *flip-flops* JK con reloj.

Suponga que utilizamos la misma asignación de estados. La tabla de transición no varía; por conveniencia, la reproducimos en la figura 8.23a.

Utilizamos la tabla de entrada del *flip-flop* de la figura 8.22d para obtener las tablas de excitación de la figura 8.23b. Se destaca una transición de estado en la tabla de transición y en la entrada correspondiente de la tabla de excitación a saber, la transición de y_2 en la figura 8.23a, de 1 a 0, requiere $J_2 = d$ y $K_2 = 1$, como muestra la figura 8.23b. A continuación, transformamos las tablas de excitación en mapas K de excitación y minimizamos las ecuaciones de lógica booleana como sigue:

$$J_1 = xy_2 \qquad K_1 = \bar{x}y_2$$
$$J_2 = x \oplus y_1 \qquad K_2 = x \odot y_1 = \bar{J}_2$$

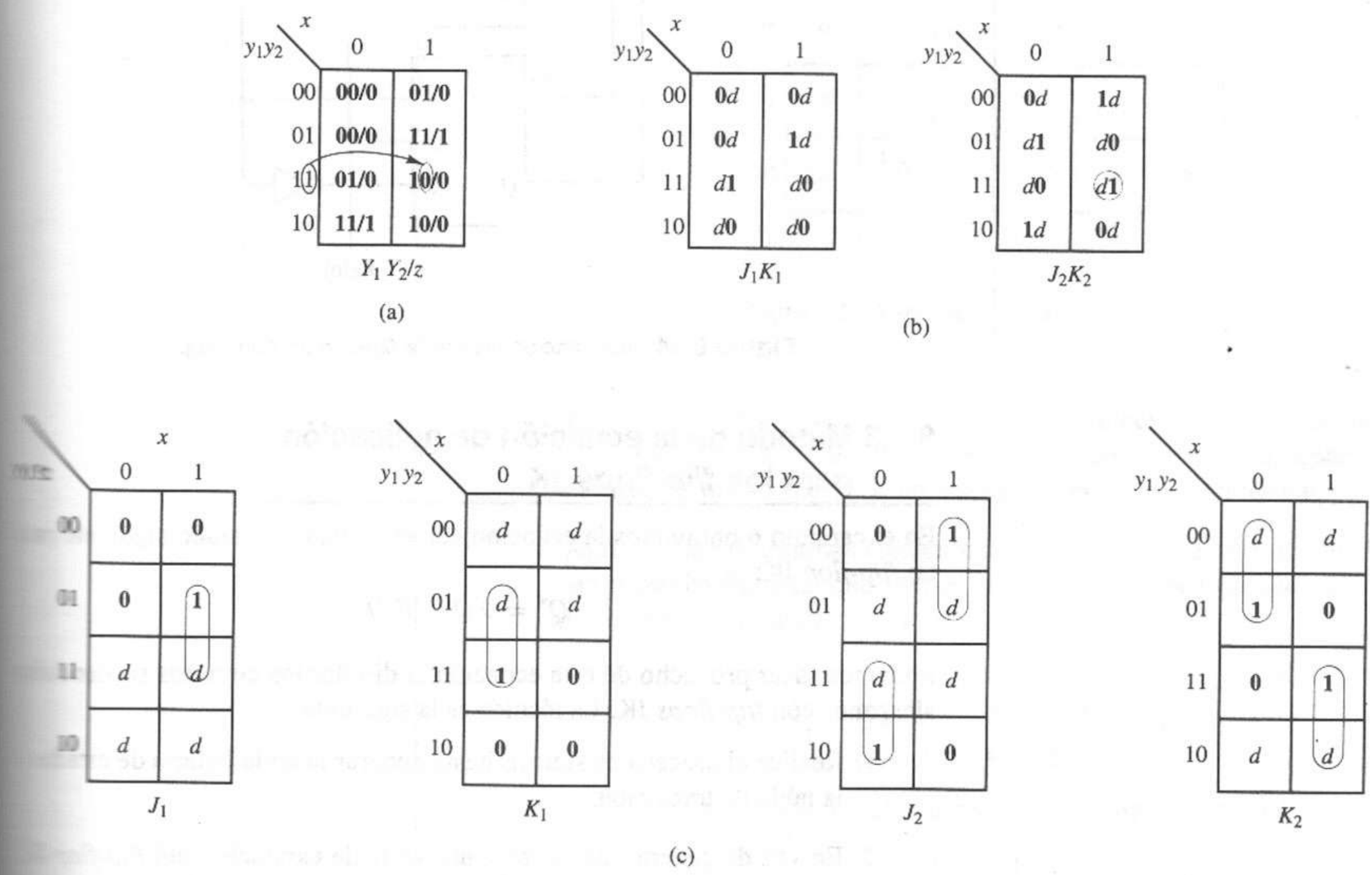


Figura 8.23 Generación de los mapas de excitación. **(a)** Tabla de transición. **(b)** Tablas de excitación. **(c)** Mapas de excitación.

La figura 8.24 muestra el diagrama lógico. La lógica de salida es idéntica a la que aparece en la figura 8.21f.

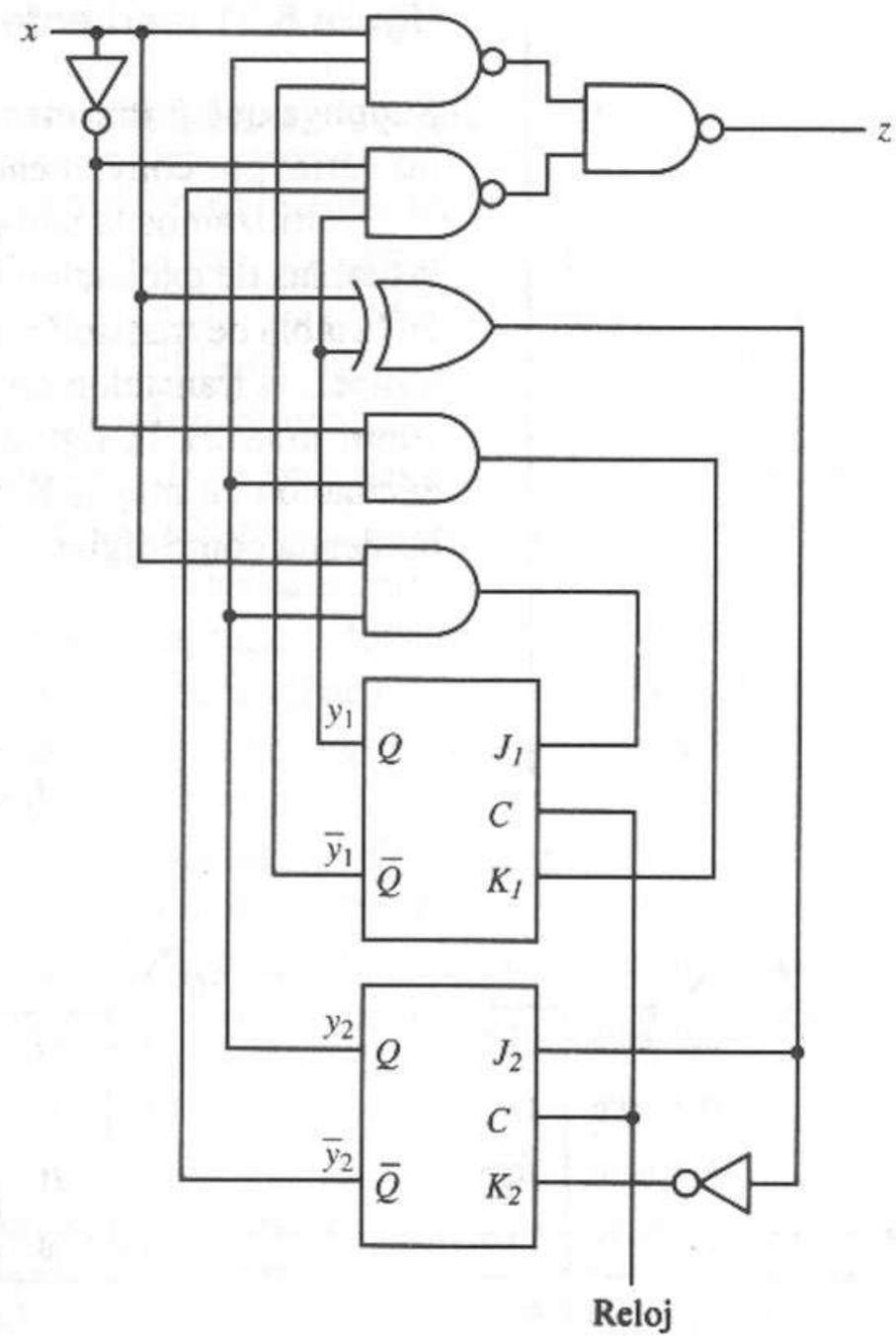


Figura 8.24 Implantación mediante *flip-flop* JK con reloj.

8.3.3 Método de la ecuación de aplicación para los *flip-flops* JK

En el capítulo 6 obtuvimos la ecuación característica del estado siguiente para un *flip-flop* JK:

$$Q^* = J\bar{Q} + \bar{K}Q$$

Podemos sacar provecho de esta ecuación al diseñar los circuitos secuenciales síncronos con *flip-flops* JK. La técnica es la siguiente:

1. Realice el proceso de síntesis hasta generar la tabla binaria de estados o la tabla de transición.

2. En vez de generar las tablas y mapas K de excitación del *flip-flop* JK, obtenga los mapas K de excitación para una implantación *flip-flop* D.

3. Para cada variable Y_i del estado siguiente, divida su mapa K en dos mitades, una asociada con y_i y otra asociada con $\bar{y}_i$.

4. Minimice la función Yi en el mapa K, sin cruzar la frontera que divide el mapa a la mitad. Esto genera una función de la forma:

$$Y_i = (J_i)\bar{y}_i + (\bar{K}_i)y_i$$

5. Podemos escribir ahora las expresiones J_i y K_i, utilizando en forma directa la función del paso 4.

EJEMPLO 8.8

Deducir las ecuaciones de excitación para el ejemplo anterior con el método de la ecuación de aplicación.

Realizamos los pasos del método de la ecuación de aplicación como sigue:

1. La tabla de transición aparece en la figura 8.23a.
2. Deducimos los mapas de excitación a partir de la tabla de transición, como se muestra en la figura 8.25.

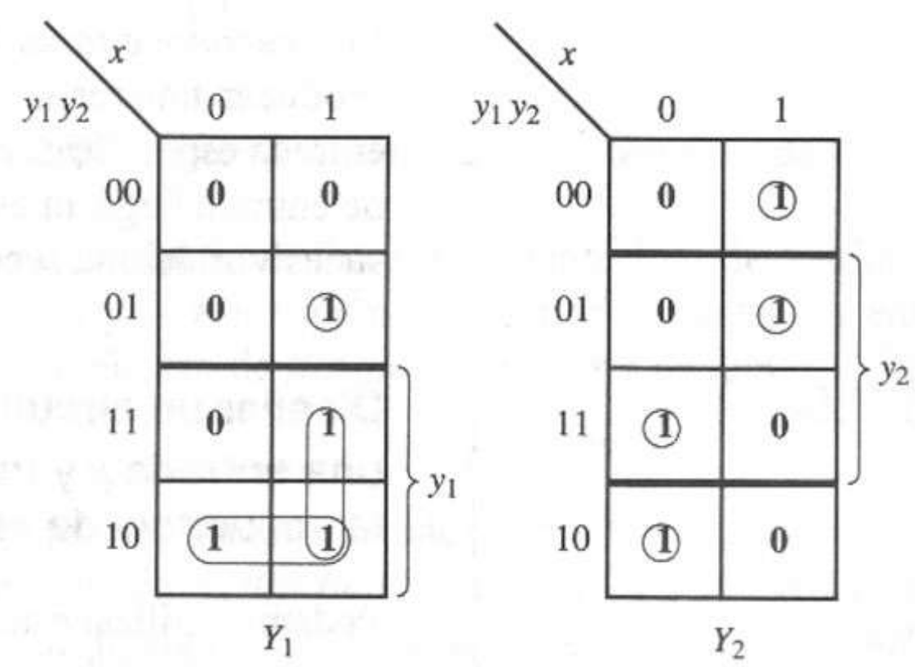


Figura 8.25 Mapas K de excitación.

3. Dividimos mediante una línea el mapa de excitación de Y_1 de la figura 8.25 en dos partes, correspondientes a y_1 y $\bar{y}_1$. De igual manera, separamos el mapa de excitación de Y_2 en dos partes, correspondientes a y_2 y $\bar{y}_2$.
4. Agrupamos los términos mediante círculos en los mapas de excitación, sin cruzar las fronteras que dividen los mapas. Esto da lugar a las siguientes expresiones lógicas:

$$\begin{aligned} Y_1 &= (xy_2)\bar{y}_1 + (x + \bar{y}_2)y_1 \\ Y_2 &= (x\bar{y}_1 + \bar{x}y_1)\bar{y}_2 + (x\bar{y}_1 + \bar{x}y_1)y_2 \\ &= (x \oplus y_1)\bar{y}_2 + (x \oplus y_1)y_2 \end{aligned}$$

5. Como las expresiones obtenidas en el paso 4 son de la forma

$$Y_i = (J_i)\bar{y}_i + (\bar{K}_i)y_i$$

podemos escribir las expresiones J_i y K_i directamente de ellas:

$$\begin{aligned} J_1 &= xy_2 & K_1 &= \overline{x + \bar{y}_2} = \bar{x}y_2 \\ J_2 &= x \oplus y_1 & K_2 &= \overline{x \oplus y_1} = x \odot y_1 \end{aligned}$$

8.3.4 Ejemplos de diseño

En este capítulo hemos presentado el procedimiento de síntesis para circuitos secuenciales síncronos como un proceso de seis pasos. El lector ya debe ser capaz de completar todos los pasos, excepto los pasos 2 y 3, la reducción de estados y la asignación de estados. Pospondremos el desarrollo de la teoría para estos temas y, en vez de ello, nos dedicaremos a ciertas aplicaciones prácticas de los otros cuatro pasos. Este cambio de ritmo tiene dos propósitos: primero, dejar que el lector practique su habilidad para formular el diagrama o tabla de estados y generar el mapa de excitación y, segundo, proporcionar una base para presentar los problemas de la reducción de la tabla de estados y de la asignación de estados.

Mostraremos el procedimiento de diseño mediante varios ejemplos. En cada uno, suponemos que el circuito es controlado por un pulso de reloj periódico y que las transiciones en los circuitos sólo ocurren iniciadas por este reloj.

Reconocedores de secuencia

Los *reconocedores de secuencia* son circuitos secuenciales síncronos que producen una respuesta dada en sus salidas al detectar secuencias de valores de entrada específicas. Suponemos que cada conjunto de valores de una secuencia de entrada llega antes que la transición activa del reloj, de modo que los valores sucesivos de una secuencia llegan durante periodos consecutivos del reloj.

EJEMPLO 8.9

Diseñar un circuito secuencial síncrono con una entrada x y una salida z que reconozca la secuencia de entrada 01.

Podemos utilizar este circuito para reconocer una transición de 0 a 1 en la entrada x. En otras palabras, el circuito debe originar una secuencia de salida $z = 01$ si ocurre la secuencia de entrada $x = 01$. Por ejemplo, si la secuencia de entrada es

$$x = 010100000111101$$

entonces la secuencia de salida será

$$z = 010100000100001$$

El primer paso del procedimiento de diseño es la construcción de un diagrama de estados que represente el comportamiento entrada/salida descrito. La figura 8.26 muestra la construcción de este diagrama. Primero suponemos que el circuito se encuentra en algún estado inicial A y que la primera entrada es 1. Como 1 no es el primer elemento de la cadena de entrada por reconocer, el circuito permanece en el estado A y devuelve una salida $z = 0$, como en la figura 8.26a. Por otro lado, si el circuito se encuentra en el estado inicial A y la entrada es 0, como esta entrada es el primer símbolo de la cadena por reconocer, el circuito pasa a un nuevo estado B y devuelve 0 como salida, lo que se muestra en la figura 8.26b. Ahora supongamos que el circuito se encuentra en el estado B y que el símbolo de entrada es 0. Como éste no es el segundo símbolo en la secuencia 01, el circuito permanece en el estado B y devuelve una salida $z = 0$. Por último, si el circuito se encuentra en el estado B y el siguiente símbolo de entrada es 1, el circuito pasa al estado A y devuelve una salida $z = 1$. Observe que este diagrama final, que aparece en la figura 8.26d, satisface la secuencia de

entradas y salidas dada con anterioridad y, por tanto, reconoce la secuencia de entrada x = 01. La tabla de estados correspondiente al diagrama de estados final aparece en la figura 8.27a. Supondremos que esta tabla de estados contiene el número mínimo de estados posibles para este circuito.

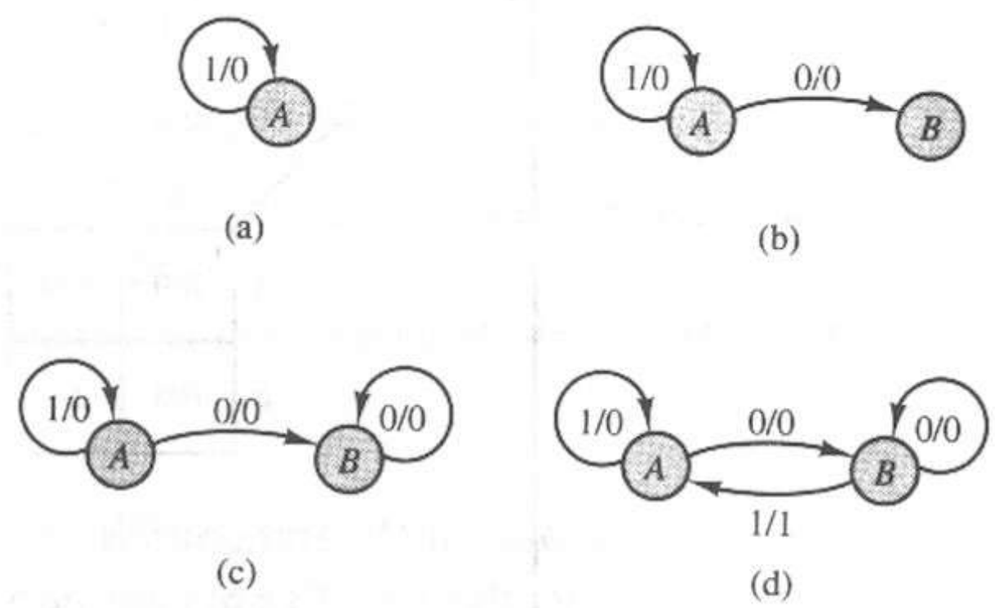


Figura 8.26 Diagrama de estados del reconocedor de la secuencia 01.

El siguiente paso del procedimiento de síntesis es determinar el número de *flip-flops* necesarios y la asignación de estados. La relación entre el número de estados (N_s) y el número de *flip-flops* (N_{FF}) está dada por la expresión

$$2^{N_{FF}-1} < N_S \leq 2^{N_{FF}} \tag{8.9}$$

Por ejemplo, un circuito de cuatro estados necesita dos *flip-flops*, un circuito de diez estados necesita cuatro *flip-flops*, etcétera. Para el circuito descrito por la tabla de estados de la figura 8.27a, sólo necesitamos un *flip-flop*. Elegimos la asignación de estados de manera arbitraria, como $A = 0$ y $B = 1$, aunque podríamos elegirla de manera opuesta ($A = 1$, $B = 0$). En el capítulo 9, daremos una explicación más detallada de la asignación de estados.

Una vez elegida la asignación de estados, podemos reescribir la tabla de estados de la figura 8.27a como la tabla de transición de la figura 8.27b. En ésta, y^k denota el estado actual del circuito, que es la salida actual del *flip-flop*. El símbolo y^{k+1} denota el estado siguiente del circuito; es decir, la salida del *flip-flop* después de ocurrir una transición. Hemos esquematizado el mapa K para la salida por separado, sólo para simplificar el diagrama. Suponga que queremos realizar el circuito con *flip-flops set-reset* con reloj. El problema se convierte entonces en el problema de determinar las señales adecuadas en las líneas de entrada *set* y *reset* para efectuar las transiciones de la figura 8.27b. Utilizamos la tabla de entrada del *flip-flop* SR con reloj de la figura 8.22b para deducir los mapas de excitación de la figura 8.27c. Por ejemplo, consideremos la transición en la esquina superior izquierda de la tabla de transición de la figura 8.27b: $y^k = 0$, $x = 0$ y $y^{k+1} = 1$. Para efectuar un cambio de estado de $y^k = 0$ a $y^{k+1} = 1$, las señales que deben aparecer en las líneas *set* y *reset* son $S = 1$ y $R = 0$. Por tanto, estas señales aparecen en las posiciones correspondientes en los mapas de excitación de la figura 8.27c. Ahora, consideremos la transición de estado de la esquina superior derecha de la tabla de transición: $y^k = 0$, $x = 1$, y $y^{k+1} = 0$. Como no debe ocurrir un cambio de estado, la señal en la línea *set* debe ser $S = 0$,

mientras que la señal en la línea *reset* no importa; es decir, R es una condición prescindible. El lector debe recordar que un *flip-flop* SR no cambia al estado *set* con $S=0$ y $R=0$ o $S=0$ y $R=1$. Los demás bloques de los mapas de excitación se determinan de manera similar.

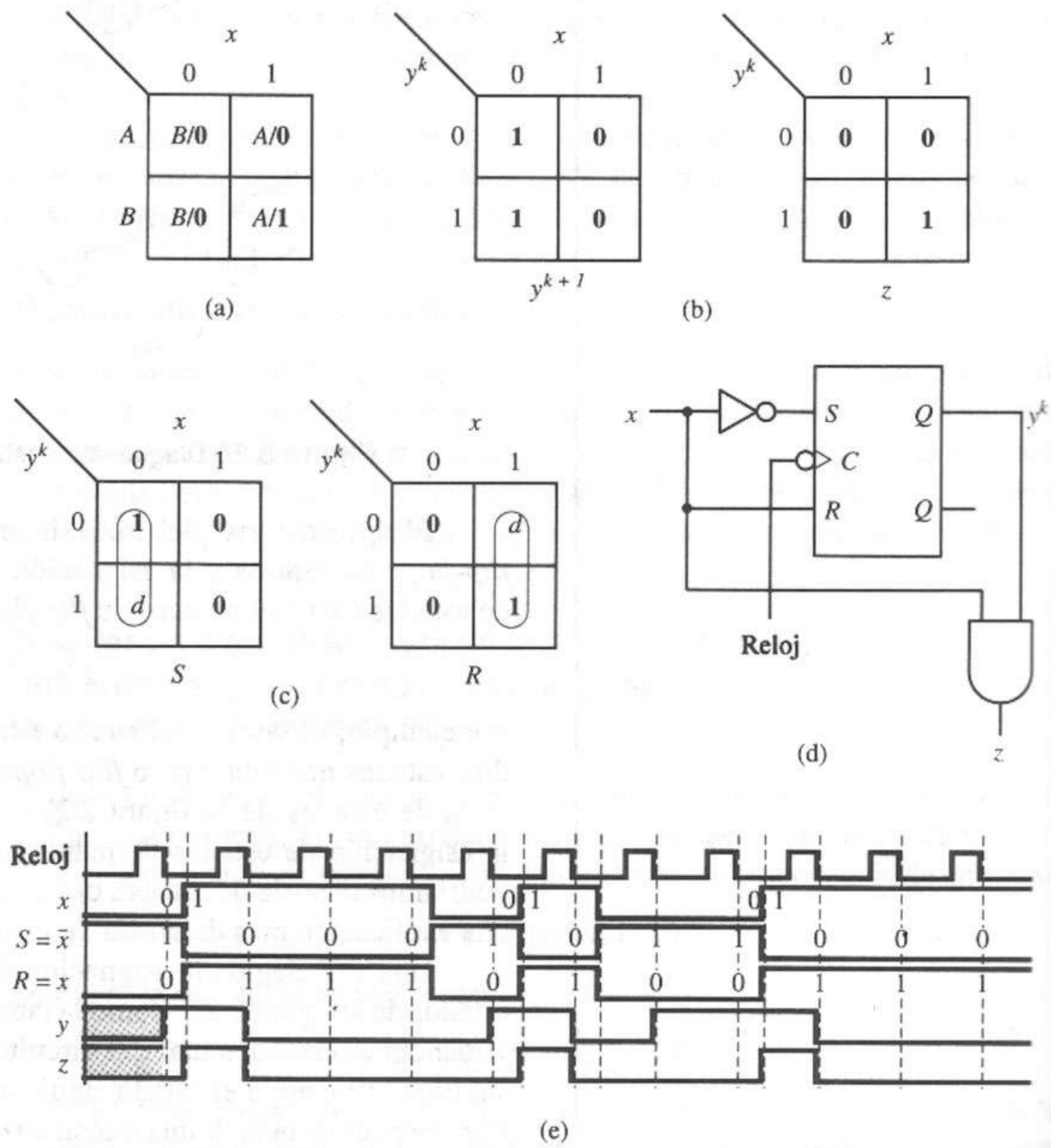


Figura 8.27 Síntesis de un reconocedor de secuencia 01. **(a)** Tabla de estados. **(b)** Tabla de transición y mapa de salida. **(c)** Mapas de excitación **(d)** Diagrama lógico. **(e)** Diagrama de tiempos.

Ahora podemos utilizar los mapas de excitación para deducir las ecuaciones del circuito lógico de conmutación:

$$S = \bar{x}$$
$$R = x$$
$$z = xy^k$$

La figura 8.27d muestra el circuito real obtenido a partir de estas ecuaciones lógicas. El lector debe verificar que el circuito realmente reconoce la secuencia

de entrada 01. La figura 8.27e muestra un diagrama de tiempos, con x como señal de entrada de nivel no sincronizada con el reloj. Observe que el circuito opera como sigue:

1. Un 0 lógico hace que el *flip-flop* SR se establezca (*set*) en la siguiente transición descendente del reloj.
2. Después, el *flip-flop* permanece en 1 lógico esperando que la entrada cambie de 0 a 1.
3. Después, x es alta y la salida z también.
4. Por último, el *flip-flop* se restablece (*reset*) en la siguiente transición descendente del reloj.

Así, hemos detectado las tres transiciones de 0 a 1 en x como muestran los tres pulsos en z.

EJEMPLO 8.10

Para completar, realicemos el circuito del ejemplo 8.9 con *flip-flops* T con reloj y después con *flip-flops* JK.

Si utilizamos un *flip-flop* T con reloj para implantar el reconocedor de secuencia, el mapa de excitación para el *flip-flop* es como se muestra en la figura 8.28a. Deducimos esta tabla a partir de la tabla de transición de la figura 8.27b. Recuerde que $T = 1$ si ocurre una transición de estado y $T = 0$ en caso contrario. La figura

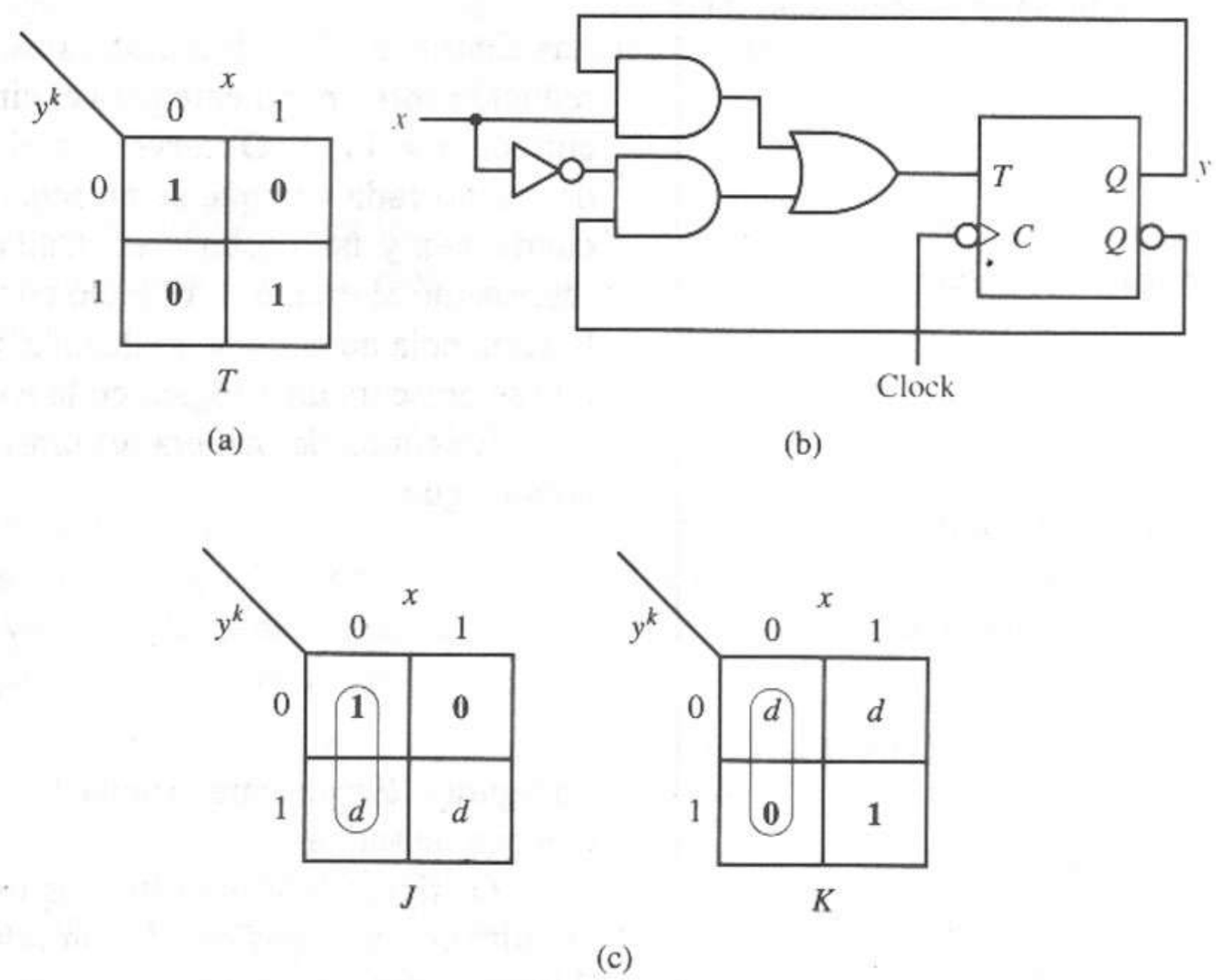


Figura 8.28 Realizaciones mediante *flip-flops* JK y T con reloj. **(a)** Mapa de excitación del *flip-flop* T con reloj. **(b)** Implantación mediante *flip-flop* T con reloj. **(c)** Mapas de excitación JK con reloj.

8.22c muestra la tabla de entrada para un *flip-flop* T con reloj. Para el mapa de excitación de la figura 8.28a, la ecuación lógica de T está dada por la expresión

$$T = \bar{x}\bar{y} + xy = x \odot y$$

La ecuación de salida es idéntica a la obtenida con anterioridad. La figura 8.28b muestra la implantación de la ecuación lógica con un *flip-flop* T.

La figura 8.28c muestra los mapas de excitación para una realización con *flip-flops* JK con reloj. Las ecuaciones lógicas correspondientes son:

$$J = \bar{x}, \quad K = x$$

Observe que las ecuaciones lógicas para J y K son idénticas a las de S y R en la realización *set-reset* y, por tanto, la realización del *flip-flop* JK con reloj es idéntica a la que aparece en la figura 8.27d, reemplazando el *flip-flop* SR por el *flip-flop* JK.

EJEMPLO 8.11

Diseñar un circuito secuencial síncrono con una línea de entrada y una línea de salida que reconozca la cadena de entrada $x = 1111$. El circuito también debe reconocer las secuencias traslapadas, como se ve en la cadena de salida z que resulte de la siguiente cadena de entrada x:

$$x = 1101111111010$$
$$z = 0000001111000$$

Las figuras 8.29a y b muestran el diagrama de estados y la tabla de estados reducida correspondiente para el circuito secuencial que reconoce la cadena de entrada $x = 1111$. Observe que si el estado inicial es A, el circuito cambia de estado cada vez que se presenta una entrada $x = 1$, con la excepción de la cuarta vez y las posteriores. Cada vez que $x = 0$, el circuito se restablece, regresando al estado A. El ciclo con $x = 1$ en el estado D satisface el criterio de la secuencia de entrada traslapada, produciendo un 1 en la salida la cuarta vez que se presenta un 1 lógico en la entrada, la quinta, la sexta, etcétera,.

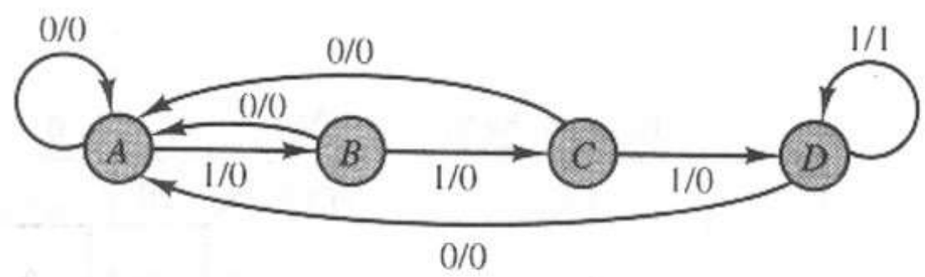
Elegimos de manera arbitraria la asignación de estados para el ejemplo como sigue:

$$A = 00$$
$$B = 01$$
$$C = 10$$
$$D = 11$$

La figura 8.29c muestra la tabla de transición resultante. La figura 8.29d muestra el mapa de salida.

La figura 8.30 muestra los mapas de excitación para la realización del circuito con un *flip-flop* SR con reloj. Las ecuaciones lógicas obtenidas de las figuras 8.29d y 8.30 son

$$S_1 = y_2 x, \quad S_2 = \bar{y}_2 x, \qquad z = x y_1 y_2$$
$$R_1 = \bar{x}, \quad R_2 = \bar{x} + \bar{y}_1 y_2$$



(a)

	x = 0	x = 1
A	$A/0$	$B/0$
B	$A/0$	$C/0$
D	$A/0$	$D/1$
C	$A/0$	$D/0$

(b)

$y_1^k y_2^k$ \ x	0	1
00	**00**	**01**
01	**00**	**10**
11	**00**	**11**
10	**00**	**11**

$y_1^{k+1} y_2^{k+1}$

(c)

$y_1^k y_2^k$ \ x	0	1
00	**0**	**0**
01	**0**	**0**
11	**0**	**1**
10	**0**	**0**

z

(d)

Figura 8.29 Reconocedor de la secuencia 1111. **(a)** Diagrama de estados. **(b)** Tabla de estados. **(c)** Tabla de transición. **(d)** Mapa de salida.

La figura 8.31 muestra los mapas de excitación para la realización del circuito con un *flip-flop* T con reloj; las ecuaciones del circuito lógico correspondiente son

$$T_1 = y_1\bar{x} + \bar{y}_1 y_2 x, \qquad z = x y_1 y_2$$
$$T_2 = y_2\bar{x} + \bar{y}_2 x + \bar{y}_1 y_2$$

La figura 8.31b muestra los mapas de excitación para la realización del circuito con un *flip-flop* JK con reloj. A continuación damos las ecuaciones lógicas obtenidas de estos mapas K; la figura 8.32 muestra el hardware utilizado para realizar estas ecuaciones.

$$J_1 = y_2 x, \quad J_2 = x, \qquad z = x y_1 y_2$$
$$K_1 = \bar{x}, \quad K_2 = \bar{y}_1 + \bar{x}$$

También podemos deducir las ecuaciones de excitación del *flip-flop* JK mediante el método de la ecuación de aplicación. Primero deducimos los mapas K de excitación a partir de la tabla de transición, como se muestra en la figura 8.31c. De ellos, podemos deducir las ecuaciones de estado siguiente.

$$Y_1 = (x y_2)\bar{y}_1 + (x) y_1$$
$$Y_2 = (x)\bar{y}_2 + (x y_1) y_2$$

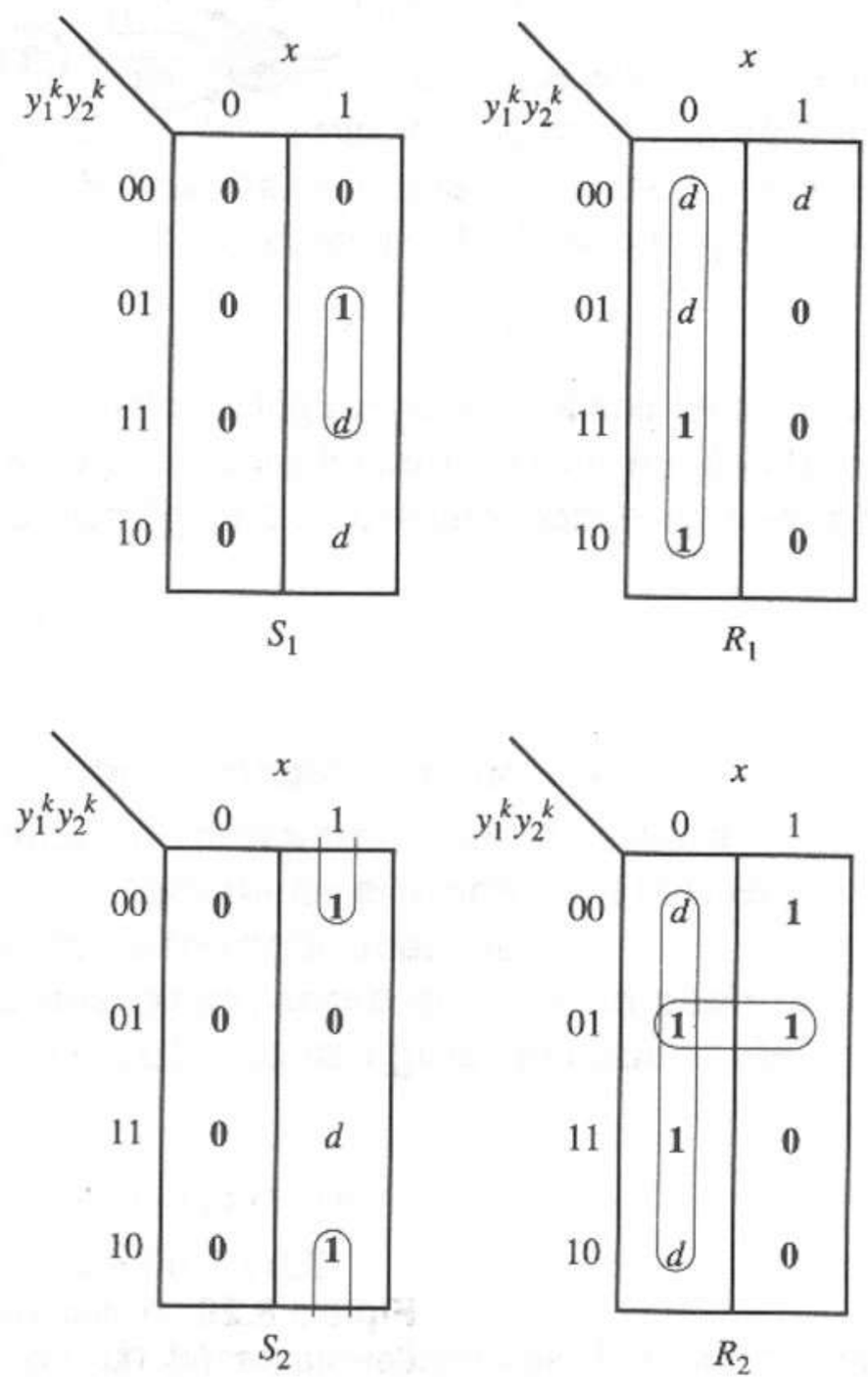


Figura 8.30 Mapas K para una realización SR con reloj.

Recordemos que las ecuaciones de excitación tienen la forma

$$Y_i = (J_i)\bar{y}_i + (\bar{K}_i)y_i$$

podemos escribir las ecuaciones J y K directamente a partir de las ecuaciones de estado siguiente, con lo que obtenemos las mismas expresiones ya deducidas de los mapas de excitación JK.

EJEMPLO 8.12

Diseñar un circuito secuencial con reloj que reconozca la secuencia de entrada que consta precisamente de dos ceros seguidos por un 10. En otras palabras, debemos obtener la siguiente secuencia de salida a partir de la secuencia de entrada dada.

$$x = 001001000010010$$
$$z = 000100100001001$$

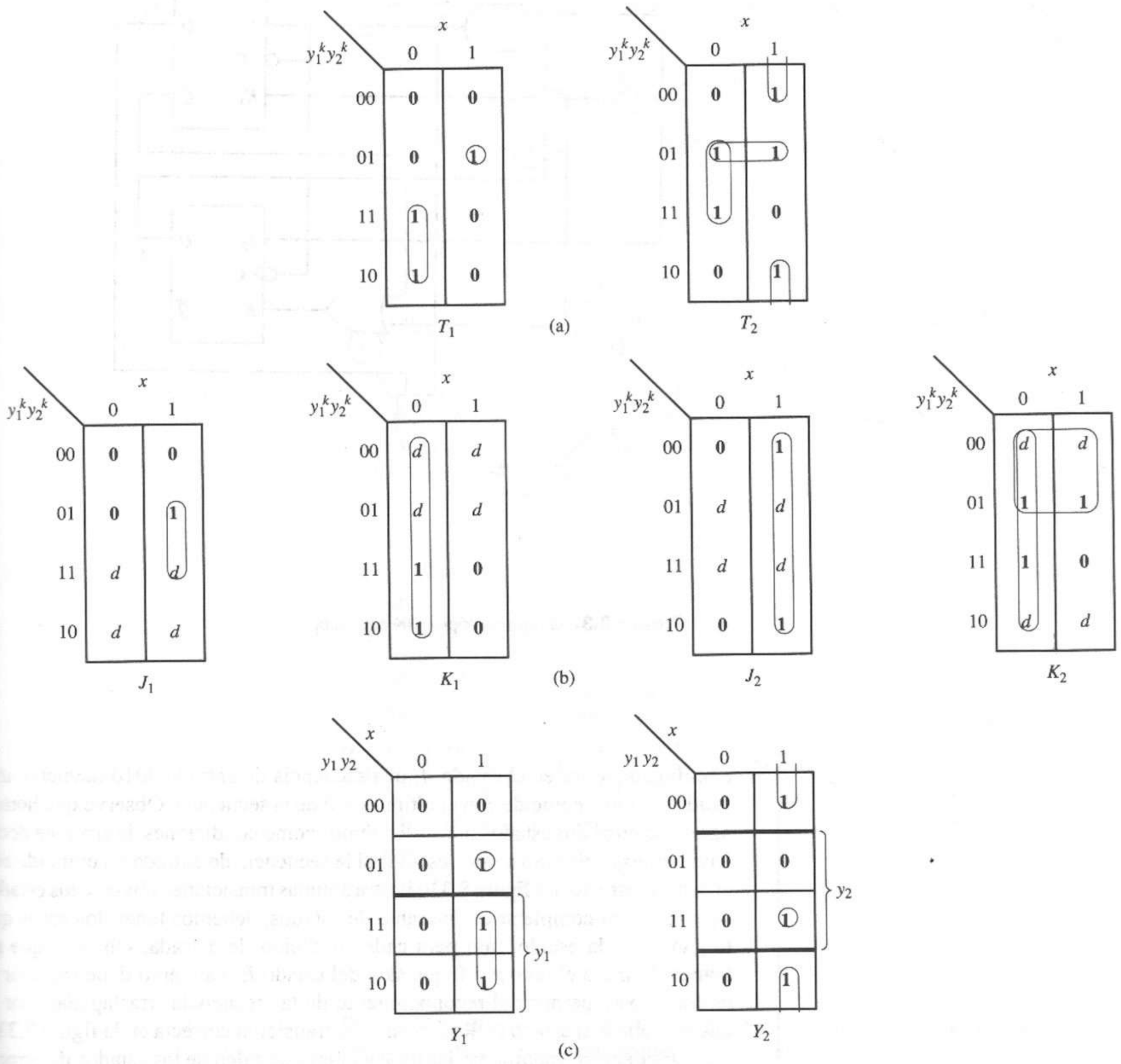


Figura 8.31 Realizaciones con *flip-flops* JK y T con reloj. **(a)** Mapas de excitación T con reloj. **(b)** Mapas de excitación JK con reloj. **(c)** Mapas K de excitación.

Una forma de enfocar el diseño de un reconocedor de secuencias es establecer una cadena de transiciones de estado correspondiente a la secuencia de entrada "correcta". Examinemos el diagrama parcial de estados de la figura 8.33a. Si el

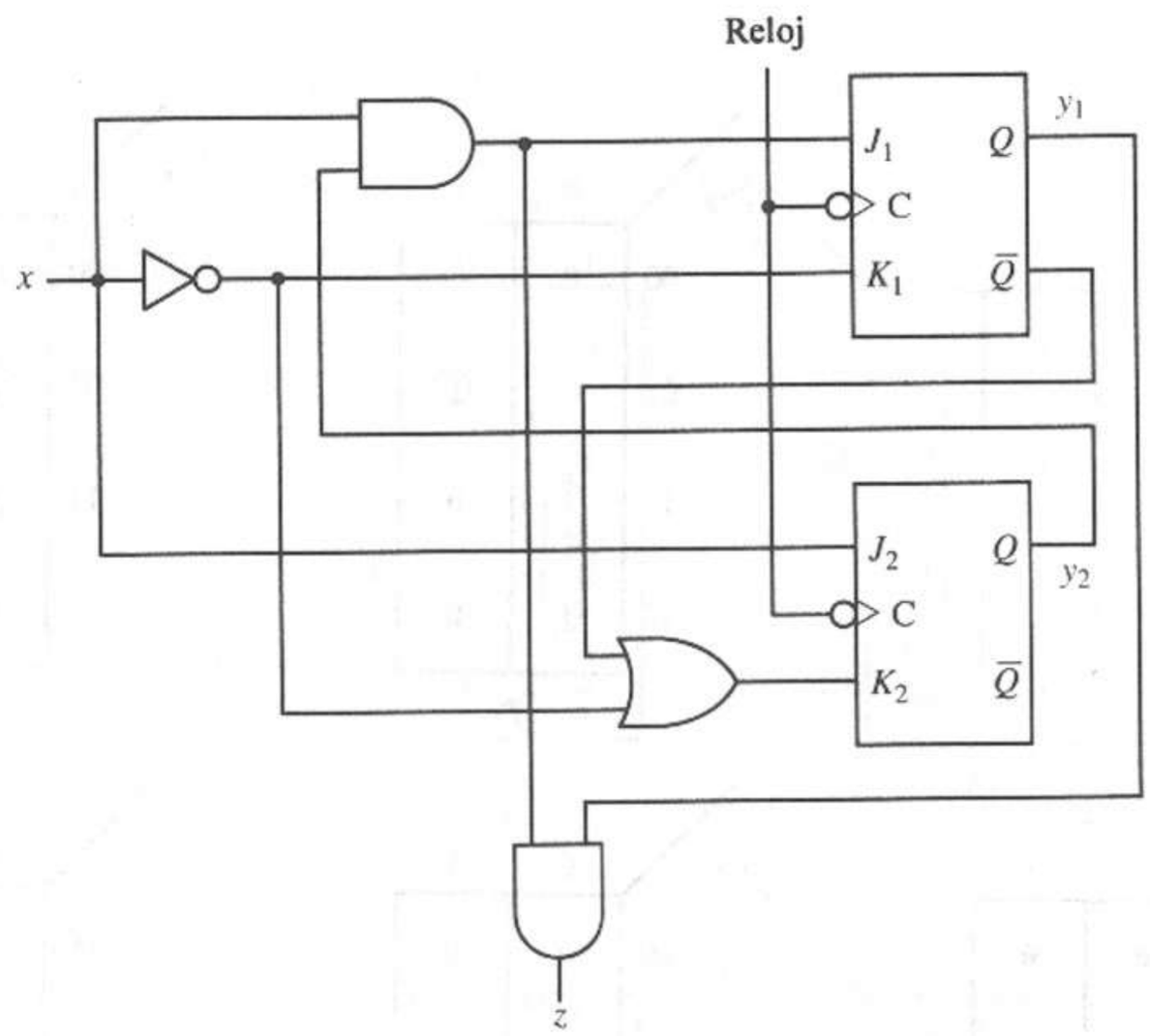


Figura 8.32 Diagrama lógico JK con reloj.

circuito comienza en el estado A, una secuencia de entrada 0010 devuelve una salida $z = 1$ que coincide con el último $x = 0$ de la secuencia. Observe que hemos agregado otros dos estados que utilizaremos como condiciones de error; es decir, enviaremos el circuito a estos estados si la secuencia de entrada no coincide con el patrón correcto. La figura 8.33b ilustra algunas transiciones obvias a los estados de error. Para completar el diagrama de estados, debemos tener dos arcos que salgan de cada estado, uno para cada condición de entrada. Observe que no hemos definido el arco $x = 0$ que sale del estado E. Este arco debe regresar al estado C para permitir el reconocimiento de las secuencias traslapadas, por lo que *no* debe ir al estado G. Registramos la transición correcta en la figura 8.33c. Después debemos completar las transiciones que salen de los estados de error F y G. Para el estado F, si se presenta una entrada $x = 0$, entonces existe un patrón de secuencia 10, que puede ser el inicio de una secuencia de entrada válida, por lo que enviamos el circuito al estado B en la fila horizontal de estados del diagrama de estados. Las demás entradas x al circuito en los estados F y G causan transiciones dentro de estos estados de error, como vemos en la figura 8.33d. Esto completa el diagrama de estados. Ahora debemos buscar los estados equivalentes en nuestro diseño. Al examinar la tabla de estados para el circuito de la figura 8.33e, observamos que $A = F$ y $B = E$, pues tienen filas idénticas, por lo que podemos eliminar las filas E y F. Así, las figuras 8.33f y g muestran el circuito reducido. En el capítulo 9 hablaremos más acerca de los estados equivalentes y la reducción de la tabla de estados.

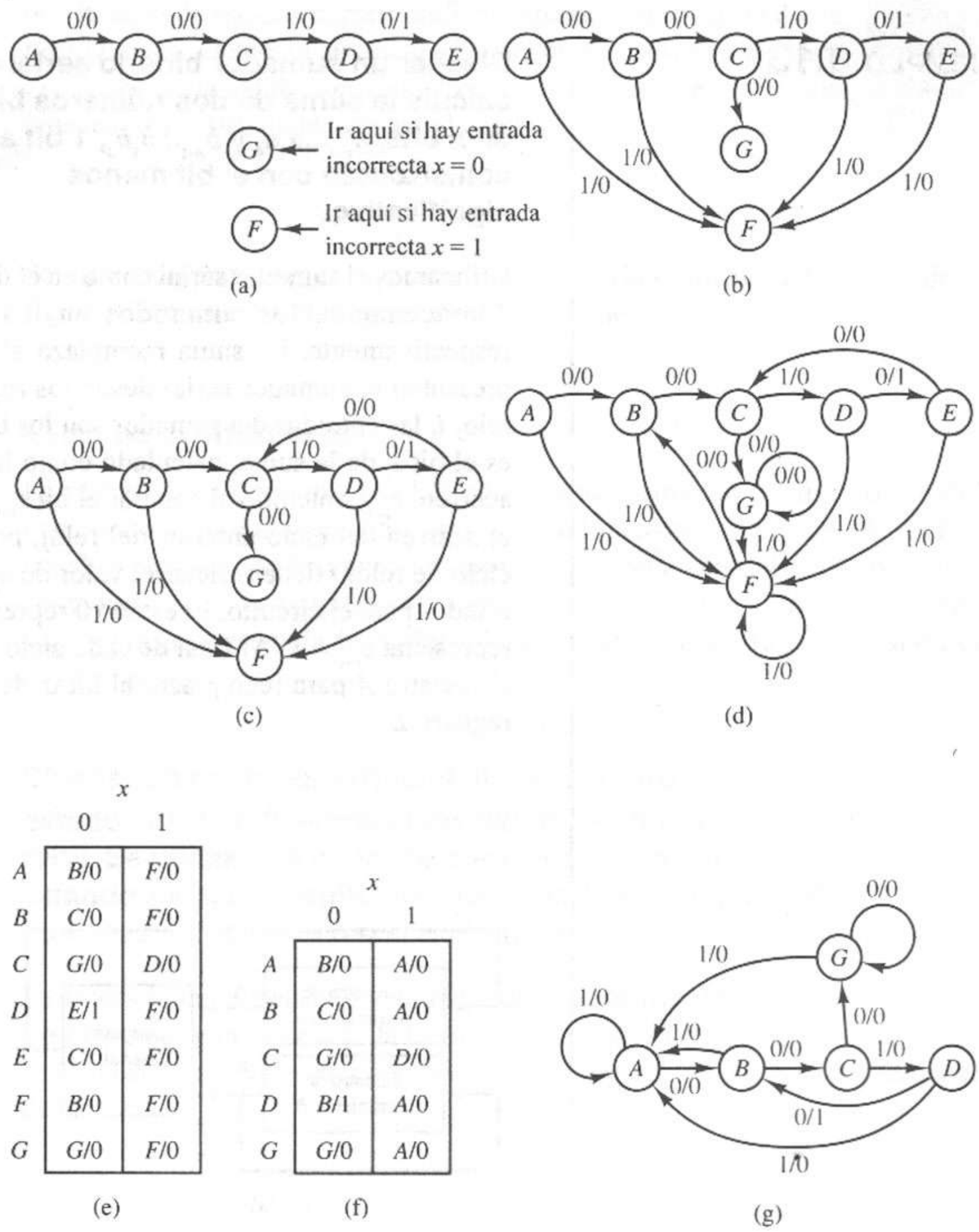


Figura 8.33 Reconocedor de la secuencia 0010. **(a)** Diagrama parcial de estados. **(b)** Transiciones de estado a los estados de error. **(c)** Retroalimentación para las secuencias traslapadas. **(d)** Diagrama de estados completo. **(e)** Tabla de estados. **(f)** Tabla de estados reducida. **(g)** Diagrama de estados.

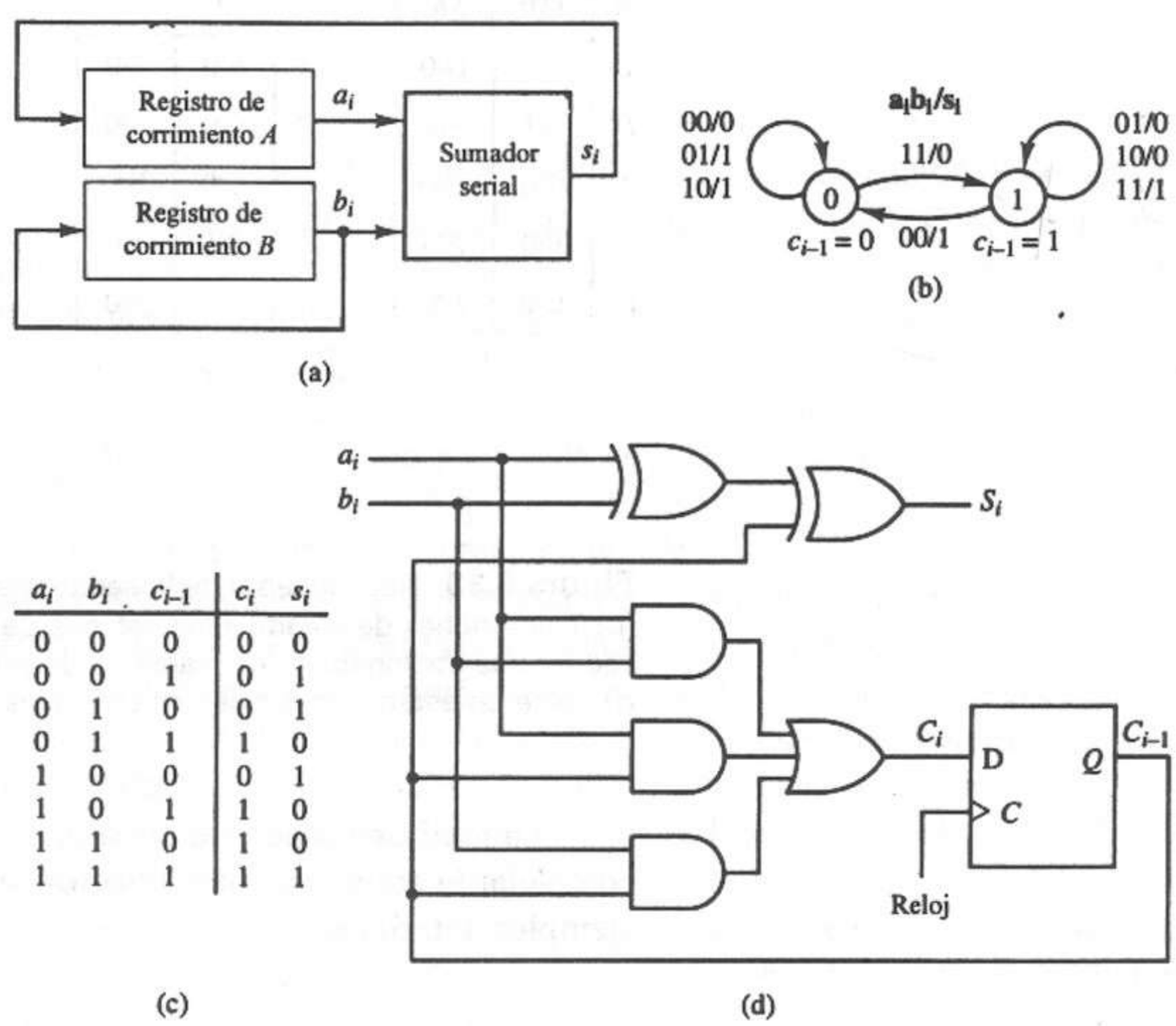
La realización de este circuito en hardware requiere tres *flip-flops*. Para completar la solución, sólo debemos seguir el procedimiento mostrado en los ejemplos anteriores.

Muchos problemas pueden resolverse de manera análoga al caso de los reconocedores de secuencias. Mostramos esto en el siguiente ejemplo, donde diseñamos un circuito aritmético que opera sobre secuencias de dígitos binarios.

EJEMPLO 8.13

Diseñar un sumador binario serial que calcule la suma de dos números binarios de n bits, $a_{n-1}...a_1a_0$ y $b_{n-1}...b_1b_0$, 1 bit a la vez, comenzando con el bit menos significativo.

Utilizamos el sumador serial como en el diagrama de bloques de la figura 8.34a. Almacenamos los sumandos en los registros de corrimiento A y B, respectivamente. La suma reemplaza al sumando del registro A. Los bits se presentan al sumador serial desde los registros de corrimiento. En el ciclo de reloj i, las entradas del sumador son los bits a_i y b_i de los sumandos, y la salida es el bit s_i de la suma, calculado como la suma de las dos entradas y el bit de acarreo, c_{i-1}, obtenido al generar el bit s_{i-1} de la suma. El circuito debe recordar el acarreo del ciclo anterior del reloj, por tanto, el estado del sumador en un ciclo de reloj i debe reflejar el valor de c_{i-1}. En consecuencia, necesitamos dos estados para el circuito. El estado 0 representa la condición $c_{i-1} = 0$ y el estado 1 representa $c_{i-1} = 1$. Al final de cada ciclo de reloj, el bit s_i de la suma se recorre al registro A para reemplazar al bit a_i del sumando, mientras que b_i regresa al registro B.



a_i	b_i	c_{i-1}	c_i	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figura 8.34 Diseño del sumador binario serial. **(a)** Diagrama de bloques. **(b)** Diagrama de estados. **(c)** Tabla de estados. **(d)** Diagrama lógico.

La figura 8.34c muestra el diagrama de estados del sumador serial, mientras que la tabla de estados correspondiente aparece en la figura 8.34c. Con un *flip-flop* D como elemento de memoria, podemos mostrar que las ecuaciones de excitación y de salida son simplemente las ecuaciones del sumador completo (ecuaciones 4.19), deducidas en el capítulo 4.

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$
$$D = c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$$

donde c_{i-1} representa el estado actual y c_i el estado siguiente del controlador. La figura 8.34d muestra el diagrama lógico completo.

Circuitos contadores

En el capítulo 7 presentamos el diseño y funcionamiento de diversos módulos contadores. Sin embargo, utilizamos procedimientos *ad hoc* en el diseño de aquellos módulos. En los siguientes ejemplos, consideramos el diseño de circuitos contadores con los métodos presentados en este capítulo. En estos ejemplos, suponemos la existencia de modelos Moore, donde el estado y la salida reflejan el valor actual del contador.

EJEMPLO 8.14

Diseñar un contador arriba/abajo con cuatro estados (0, 1, 2, 3) empleando *flip-flops* JK con reloj. Se utiliza una señal de control x como sigue: Cuando $x = 0$, el circuito cuenta hacia adelante (arriba); cuando $x = 1$, cuenta hacia atrás (abajo).

La figura 8.35a muestra un diagrama de estados que describe a este contador. De este diagrama se deduce la tabla de estados de la figura 8.35b. Observe que la salida del contador es precisamente su estado actual. Si elegimos una asignación de estados

$$0 \rightarrow 00$$
$$1 \rightarrow 01$$
$$2 \rightarrow 10$$
$$3 \rightarrow 11$$

que es estándar para los contadores, podemos obtener la tabla de transición de la figura 8.35c. Utilizamos la tabla de entradas del *flip-flop* JK con reloj (ver la figura 8.22), para obtener los mapas de excitación para los *flip-flops* y_1 y y_2 de la figura 8.35d. Con estos mapas K, encontramos las siguientes relaciones:

$$J_1 = K_1 = x\bar{y}_2 + \bar{x}y_2 = x \oplus y_2$$
$$J_2 = K_2 = 1$$

Por tanto, obtenemos el diagrama lógico para el contador de cuatro estados arriba/abajo que se muestra en la figura 8.36. Si la señal x es controlada por un conmutador alternante y el periodo del reloj es muy lento (digamos 1 segundo), podemos observar la acción de este dispositivo colocando diodos emisores de luz (LED) en las salidas de los *flip-flops*.

Ahora veremos un diseño de contador más complejo.

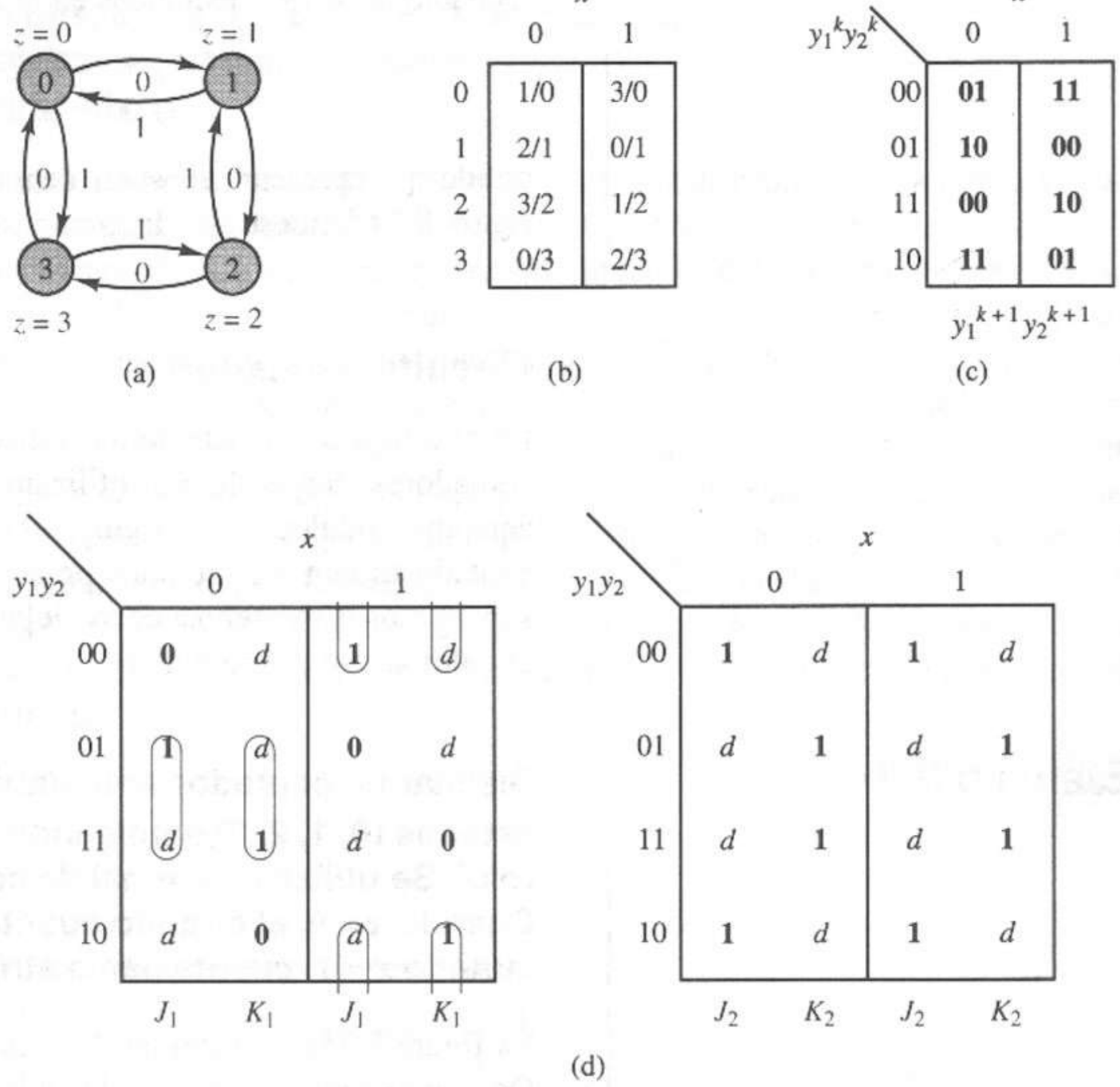


Figura 8.35 Síntesis del contador arriba/abajo. **(a)** Diagrama de estados. **(b)** Tabla de estados. **(c)** Tabla de transición. **(d)** Mapas de excitación.

EJEMPLO 8.15

Utilizar *flip-flops* JK con reloj para diseñar un circuito que cuente en el código BCD. El contador tiene una señal de control, x. Cuando $x = 1$, el contador cuenta; de otro modo, retiene el estado actual. La salida que muestra el valor del contador tiene la forma de cuatro luces. Por ejemplo, si el contador es 3, entonces las luces estarían OFF, OFF, ON, ON.

Debido a su sencillez, podemos construir la tabla de estados de inmediato. Ignoramos la salida por el momento y mostramos la tabla de estados para el ejemplo en la figura 8.37a.

A fin de satisfacer el requisito de la lectura de salida, asignamos los estados de modo que sean una indicación directa de conteo; es decir, cada uno recibe su representación BCD, así que podemos obtener la salida vigilando las salidas de

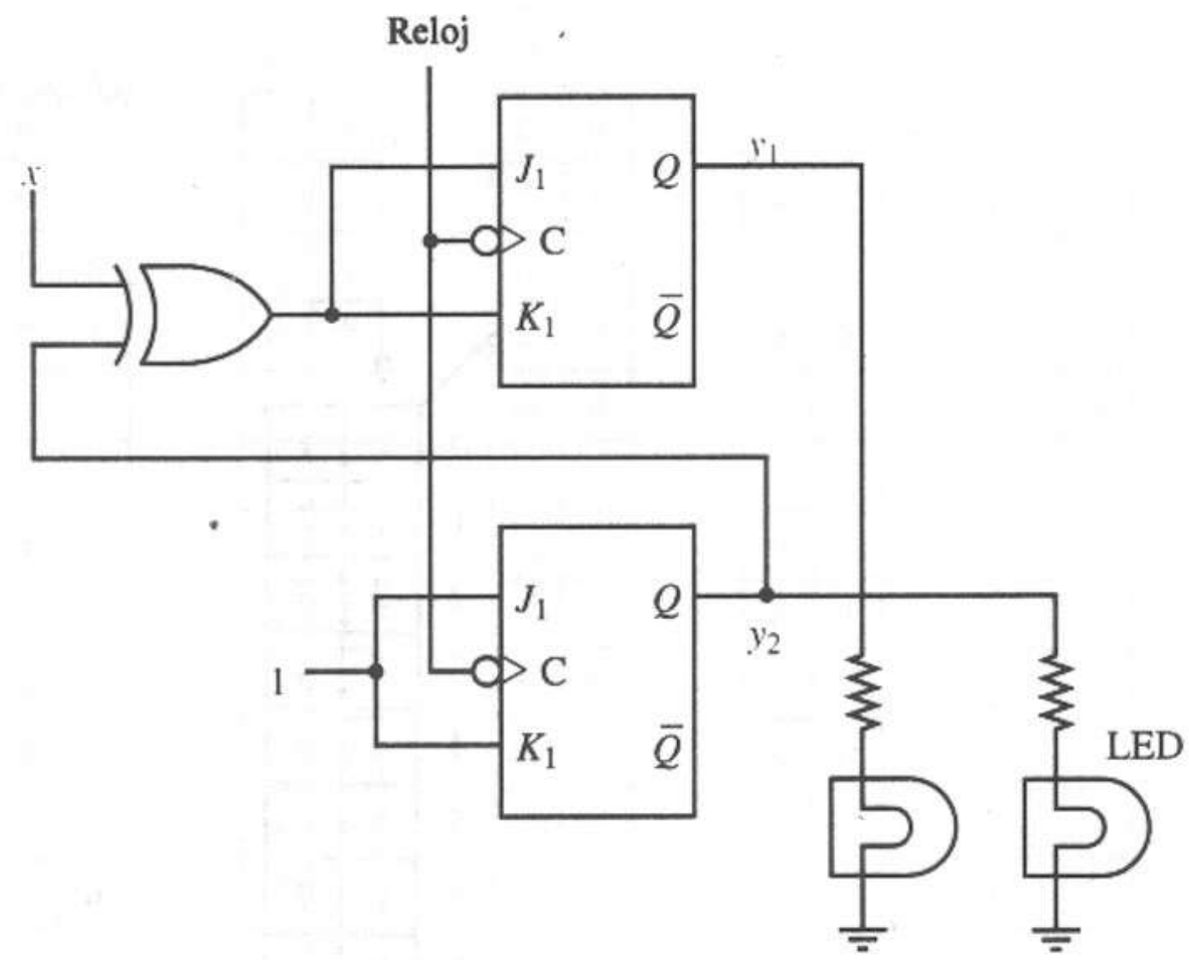


Figura 8.36 Implantación de un contador arriba/abajo.

los *flip-flops* y utilizando estas señales para encender y apagar las luces. Por tanto, la asignación de estados es

$$\begin{array}{ll} 0 \rightarrow 0000 & 5 \rightarrow 0101 \\ 1 \rightarrow 0001 & 6 \rightarrow 0110 \\ 2 \rightarrow 0010 & 7 \rightarrow 0111 \\ 3 \rightarrow 0011 & 8 \rightarrow 1000 \\ 4 \rightarrow 0100 & 9 \rightarrow 1001 \end{array}$$

La tabla de transición para esta asignación aparece en la figura 8.37b. Después, deducimos los mapas de excitación para los cuatro *flip-flops* mediante la figura 8.22d. Presentamos las tablas resultantes en la figura 8.37c. Las letras *d* en las tablas representan estados prescindibles. Podemos reorganizar estas tablas en forma de mapa K, como se muestra para la entrada J_2 de la figura 8.37d. Las ecuaciones lógicas booleanas resultantes son

$$\begin{array}{ll} J_3 = y_2 y_1 y_0 x, & K_3 = y_0 x \\ J_2 = y_1 y_0 x, & K_2 = y_1 y_0 x \\ J_1 = \bar{y}_3 y_0 x, & K_1 = y_0 x \\ J_0 = x, & K_0 = x \end{array}$$

Observe de las ecuaciones lógicas que la entrada x actúa como una señal de compuerta para activar o desactivar todas las entradas del *flip-flop* de manera simultánea.

La figura 8.37e muestra la implantación real del circuito mediante *flip-flops* JK con reloj.

	x = 0	x = 1
0	0	1
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9
9	9	0

(a)

$y_3^k y_2^k y_1^k y_0^k$	x = 0	x = 1
0000	**0000**	**0001**
0001	**0001**	**0010**
0010	**0010**	**0011**
0011	**0011**	**0100**
0100	**0100**	**0101**
0101	**0101**	**0110**
0110	**0110**	**0111**
0111	**0111**	**1000**
1000	**1000**	**1001**
1001	**1001**	**0000**
1010	*dddd*	*dddd*
1011	*dddd*	*dddd*
1100	*dddd*	*dddd*
1101	*dddd*	*dddd*
1110	*dddd*	*dddd*
1111	*dddd*	*dddd*

$y_3^{k+1} y_2^{k+1} y_1^{k+1} y_0^{k+1}$

(b)

Figura 8.37 Diseño del contador BCD. **(a)** Tabla de estados. **(b)** Tabla de transición.

EJEMPLO 8.16

Utilizar el método de la ecuación de aplicación para deducir las ecuaciones de excitación para el *flip-flop* y_1 del contador BCD del ejemplo 8.15.

1. La tabla de transición aparece en la figura 8.37b.
2. El mapa K de excitación para el *flip-flop* D y_1 aparece en la figura 8.38.
3. La línea gruesa en el mapa K divide Y_1 en mitades asociadas con y_1 y $\bar{y}_1$.
4. La ecuación del estado siguiente es

$$Y_1 = (x\bar{y}_3 y_0)\bar{y}_1 + (\bar{x} + \bar{y}_0)y_1$$

$y_3^k y_2^k y_1^k y_0^k$ \ x	J_3		K_3		J_2		K_2		J_1		K_1		J_0		K_0	
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0000	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	**0**	**1**	*d*	*d*
0001	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	**0**	**1**	*d*	*d*	*d*	*d*	**0**	**1**
0010	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	*d*	*d*	**0**	**0**	**0**	**1**	*d*	*d*
0011	**0**	**0**	*d*	*d*	**0**	**1**	*d*	*d*	*d*	*d*	**0**	**1**	*d*	*d*	**0**	**1**
0100	**0**	**0**	*d*	*d*	*d*	*d*	**0**	**0**	**0**	**0**	*d*	*d*	**0**	**1**	*d*	*d*
0101	**0**	**0**	*d*	*d*	*d*	*d*	**0**	**0**	**0**	**1**	*d*	*d*	*d*	*d*	**0**	**1**
0110	**0**	**0**	*d*	*d*	*d*	*d*	**0**	**0**	*d*	*d*	**0**	**0**	**0**	**1**	*d*	*d*
0111	**0**	**1**	*d*	*d*	*d*	*d*	**0**	**1**	*d*	*d*	**0**	**1**	*d*	*d*	**0**	**1**
1000	*d*	*d*	**0**	**0**	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	**0**	**1**	*d*	*d*
1001	*d*	*d*	**0**	**1**	**0**	**0**	*d*	*d*	**0**	**0**	*d*	*d*	*d*	*d*	**0**	**1**
1010	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
1011	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
1100	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
1101	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
1110	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
1111	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*

(c)

Figura 8.37 (continuación) Diseño del contador BCD. **(c)** Tablas de excitación para el contador BCD.

Observe que los grupos de términos producto no cruzan la línea gruesa que divide el mapa K.

5. Las ecuaciones para J_1 y K_1 son

$$J_1 = x\bar{y}_3 y_0$$
$$\bar{K}_1 = \bar{x} + \bar{y}_0$$
$$K_1 = x y_0$$

Observe que éstas son las mismas ecuaciones obtenidas mediante el método de la tabla de excitación en el ejemplo 8.15.

Controladores de estado finito

Muchas aplicaciones requieren circuitos de control que realicen secuencias dadas de acciones en respuesta a señales aplicadas en forma externa o como un resultado de condiciones producidas dentro del circuito, como un acarreo generado en una operación de suma o un contador que alcanza un valor terminal. Tales

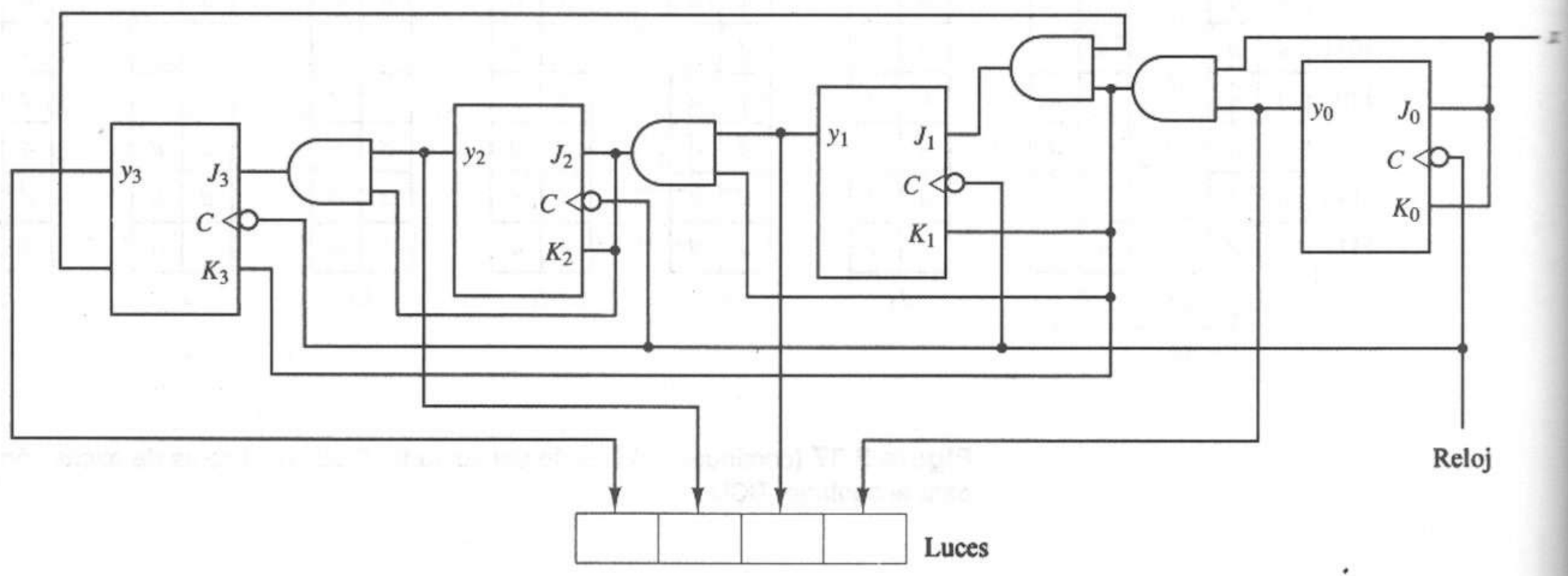
$y_1^k y_0^k$ \ $y_2^k y_3^k$	00	01	11	10
00	0	d	d	0
01	0	d	d	0
11	0	d	d	d
10	0	d	d	d

$x = 0$

00	01	11	10
0	d	d	0
0	d	d	0
1	d	d	d
0	d	d	d

$x = 1$

(d)



(e)

Figura 8.37 (continuación) Diseño del contador BCD. **(d)** Mapa K para J_2. **(e)** Realización del contador BCD.

circuitos de control se caracterizan por tener un número finito de estados; por ello, se les llama *controladores de estado finito*.

Una de las aplicaciones más comunes de los controladores de estado finito es como unidades de control para computadoras y otros sistemas digitales. Tales sistemas tienen dos partes: una ruta de datos y una unidad de control. La ruta de datos realiza varias operaciones sobre los elementos de los datos, como operaciones aritméticas y otras transformaciones. Por lo general, las rutas de datos comprenden módulos lógicos combinatorios, como las unidades lógica aritmética y los multiplexores, y con frecuencia tienen registros para el almacenamiento de datos.

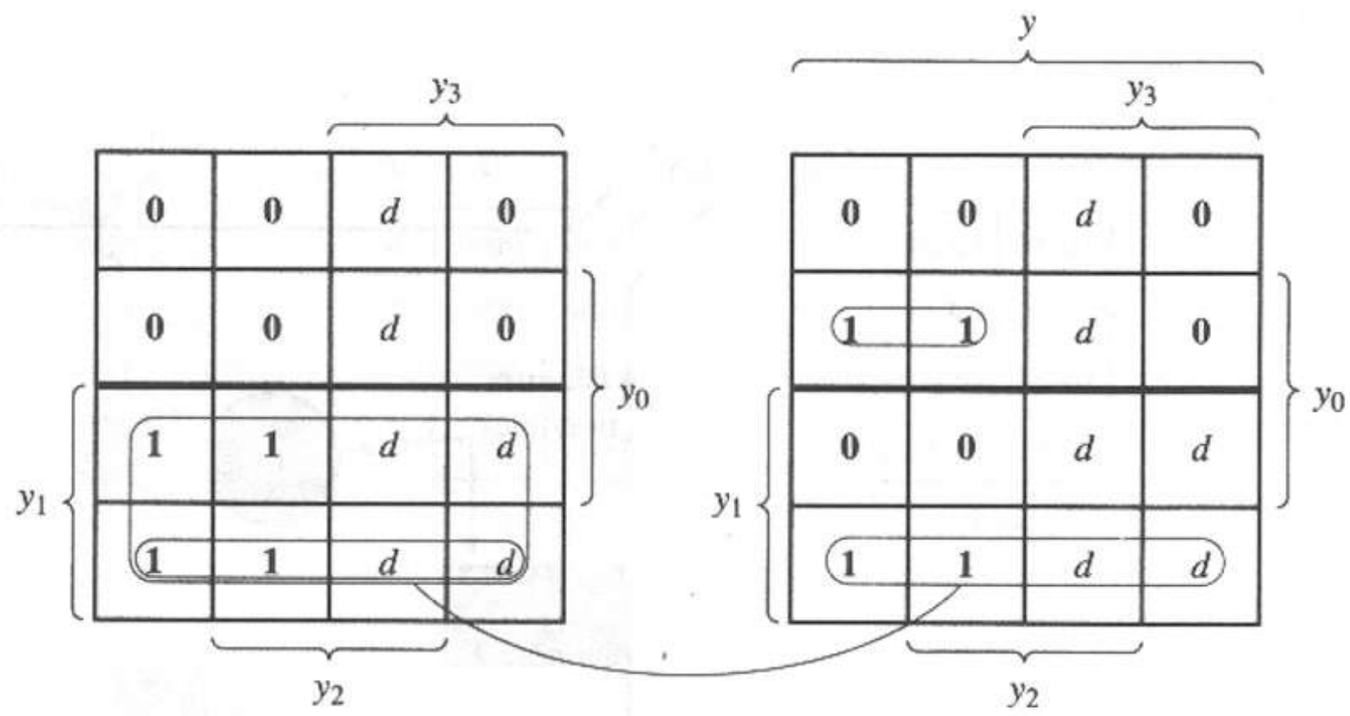


Figura 8.38 Mapa K para $Y_1(y_1^{k+1})$.

La unidad de control alimenta instrucciones a la ruta de datos para habilitar ciertas operaciones. Estas instrucciones deben tener una secuencia correcta para garantizar que se llevan a cabo las operaciones adecuadas en respuesta a las diversas entradas y condiciones. Diseñamos una unidad de control para un sistema digital identificando sus entradas y sus salidas y desarrollando después el algoritmo de control en forma de diagrama de estados.

Diseñamos los controladores de estado finito mediante el procedimiento de síntesis de un circuito secuencial síncrono ya definido. Comenzamos definiendo las entradas y salidas del controlador y después el algoritmo por implantar. Con frecuencia, especificamos el algoritmo en forma de diagrama de estados. Después realizamos los pasos restantes del procedimiento de síntesis. Mostramos esto en los siguientes ejemplos.

EJEMPLO 8.17

Diseñar un controlador de estado finito para el robot de la figura 8.39, para que encuentre la salida del laberinto de la misma figura.

El robot debe maniobrar girando cuando entre en contacto con un obstáculo. La nariz del robot tiene un sensor cuya salida es $x = 1$ siempre que se encuentra en contacto con un obstáculo; es $x = 0$ en caso contrario. El robot tiene dos líneas de control: $z_1 = 1$, que gira al robot hacia la izquierda, y $z_2 = 1$, que gira el robot hacia la derecha. Cuando encuentra un obstáculo, el robot deberá girar hacia la derecha hasta no detectar obstáculo alguno. La siguiente vez que detecte un obstáculo, el robot debe girar hacia la izquierda hasta que no haya obstáculo, y así sucesivamente.

El controlador del robot requiere los siguientes cuatro estados:

Estado A = no se detecta obstáculo, el último giro fue a la izquierda

Estado B = se detecta un obstáculo, gira a la derecha

Estado C = no se detecta obstáculo, el último giro fue a la derecha

Estado D = se detecta un obstáculo, gira a la izquierda

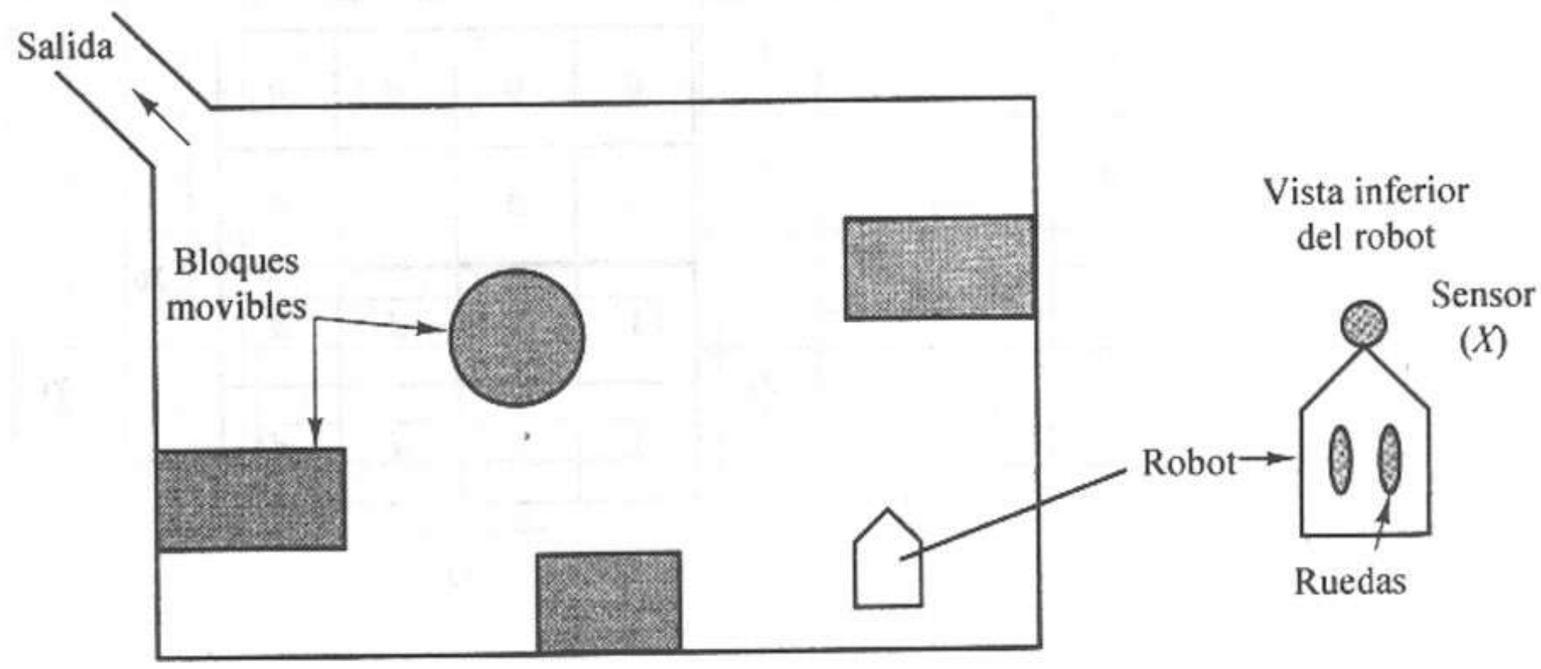


Figura 8.39 Robot y laberinto.

El diagrama de estados de la unidad de control aparece en la figura 8.40a. Observe que el controlador permanece en el estado A sin girar hasta que encuentra un obstáculo. Después entra en el estado B y gira a la derecha hasta que no detecta más obstáculos; en este momento, entra en el estado C y deja de girar. Entonces el controlador permanece en el estado C hasta que detecta otro obstáculo; en ese momento entra en el estado D y gira a la izquierda hasta no encontrar más obstáculos; regresa entonces al estado A.

La figura 8.40b muestra la tabla de estados. Elegimos la asignación de estados $A = 00$, $B = 01$, $C = 11$ y $D = 10$. La tabla binaria de transición aparece en la figura 8.40c. De la tabla de transición, podemos formar las tablas de salida para z_1 y z_2, que se muestran en la figura 8.40d, de lo cual, a su vez, podemos deducir las siguientes ecuaciones de salida.

$$z_1 = xy_1$$
$$z_2 = x\bar{y}_1$$

Para las variables de estado, utilizamos *flip-flops* JK y el método de la ecuación de aplicación. La figura 8.40e tiene las tablas de excitación para Y_1 y Y_2. De estas tablas deducimos las ecuaciones de excitación como sigue.

$$Y_1 = (\bar{x}y_2)\bar{y}_1 + (x + y_2)y_1$$
$$J_1 = \bar{x}y_2$$
$$K_1 = \overline{x + y_2} = \bar{x}\bar{y}_2$$

$$Y_2 = (x\bar{y}_1)\bar{y}_2 + (\bar{x} + \bar{y}_1)y_2$$
$$J_2 = x\bar{y}_1$$
$$K_2 = \overline{\bar{x} + \bar{y}_1} = xy_1$$

La figura 8.40f muestra el diagrama terminado para el circuito lógico del controlador del robot.

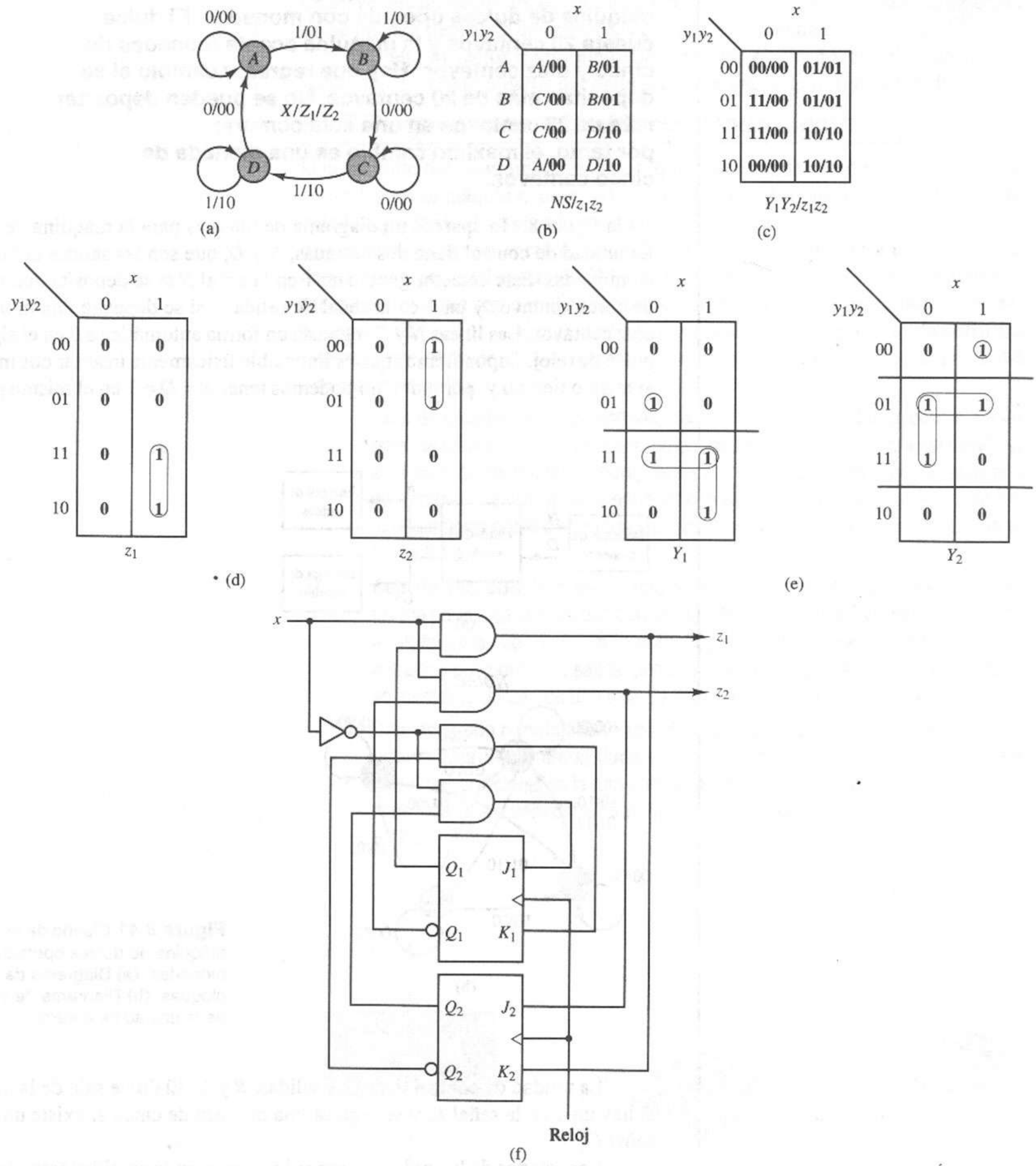
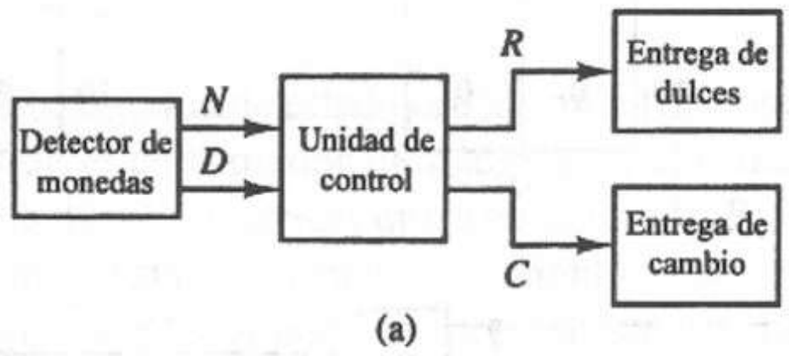


Figura 8.40 Diseño del controlador del robot. **(a)** Diagrama de estados. **(b)** Tabla de estados. **(c)** Tabla de transición. **(d)** Mapas de salida. **(e)** Mapas de excitación. **(f)** Diagrama del circuito lógico.

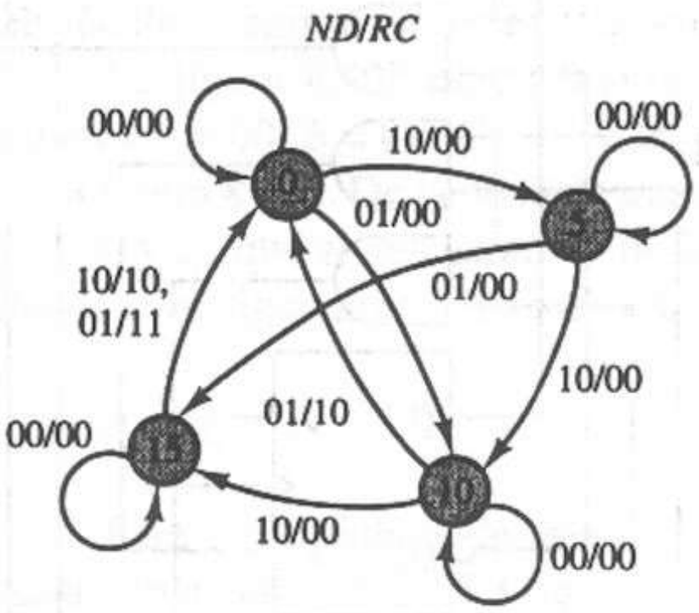
EJEMPLO 8.18

Diseñar una unidad de control para una máquina de dulces operada con monedas. El dulce cuesta 20 centavos y la máquina acepta monedas de cinco y diez centavos. Hay que regresar cambio si se depositan más de 20 centavos. No se pueden depositar más de 25 centavos en una sola compra; por tanto, el máximo cambio es una moneda de cinco centavos.

En la figura 8.41a aparece un diagrama de bloques para la máquina de dulces. La unidad de control tiene dos entradas, N y D, que son las salidas del detector de monedas. Este detector genera un 1 en la señal N si se deposita una moneda de cinco centavos y un 1 en la señal de salida D si se deposita una moneda de diez centavos. Las líneas N y D regresan en forma automática a 0 en el siguiente pulso de reloj. Supondremos que es imposible físicamente insertar dos monedas al mismo tiempo y, por tanto, no podemos tener $N = D = 1$ en el mismo periodo de reloj.



(a)



(b)

Figura 8.41 Diseño de la máquina de dulces operada con monedas. **(a)** Diagrama de bloques. **(b)** Diagrama de estados de la unidad de control.

La unidad de control tiene dos salidas, R y C. El dulce sale de la máquina si hay un 1 en la señal R, y se regresa una moneda de cinco si existe un 1 en la señal C.

Los estados de la unidad de control representan la cantidad total de dinero depositado para la compra actual. El conjunto de estados es entonces {0, 5, 10, 15}. Cuando una moneda depositada incrementa la cantidad a 20 o 25 centavos, la unidad de control regresa al estado 0 y entrega el dulce, junto con el cambio, en caso necesario.

El diagrama de estados de la figura 8.41b describe el funcionamiento de la unidad de control de la máquina de dulces. Para un estado dado, las salidas R y C dependen de la moneda insertada; por tanto, utilizamos un modelo de Mealy. Observe que no hemos especificado la combinación de entrada $ND = 11$, ya que no se pueden depositar dos monedas en forma simultánea.

Se deja al lector como ejercicio terminar el diseño del circuito lógico secuencial síncrono a partir del diagrama de estados de la figura 8.41b.

8.3.5 Diagramas de máquina de estado algorítmico

Una variante del diagrama de estados, muy útil cuando diseñamos unidades de control y otras máquinas de estados finitos, es el diagrama de la *máquina de estado algorítmico* (ASM). Un *algoritmo* es una secuencia de pasos bien definida que produce una secuencia deseada de acciones en respuesta a una secuencia dada de entradas. El diagrama ASM es una herramienta conveniente para expresar los algoritmos que serán realizados por circuitos lógicos secuenciales, así como podemos utilizar un diagrama de flujo para describir un algoritmo de software que será programado en una computadora. Construimos los diagramas ASM con tres elementos:

Caja de estado: Representa un estado del circuito y, por tanto, equivale a un nodo de un diagrama de estados. En la caja se escriben el nombre del estado y, en el caso de los circuitos Moore, las salidas por producir; es decir, las acciones por realizar mientras se encuentre en ese estado. Una caja de estado siempre tiene un único punto de entrada y un único punto de salida, como lo muestra la figura 8.42a.

Caja de decisión: Representa una decisión de transición de estado con base en una prueba de una entrada del circuito. Una caja de decisión, como la ilustrada en la figura 8.42b, enumera la entrada por verificar, tiene un único punto de entrada y dos puntos de salida, uno si la entrada es igual a 0 y otro si la entrada es igual a 1. Utilizamos una caja de decisión por cada prueba de entrada.

Caja de salida condicional: Especifica las salidas asociadas a transiciones de estado para una entrada dada en un circuito Mealy. Colocamos una caja de salida condicional en la ruta entre una caja de decisión y una caja de estado; tiene una única entrada y un punto de salida, como se muestra en la figura 8.42c.

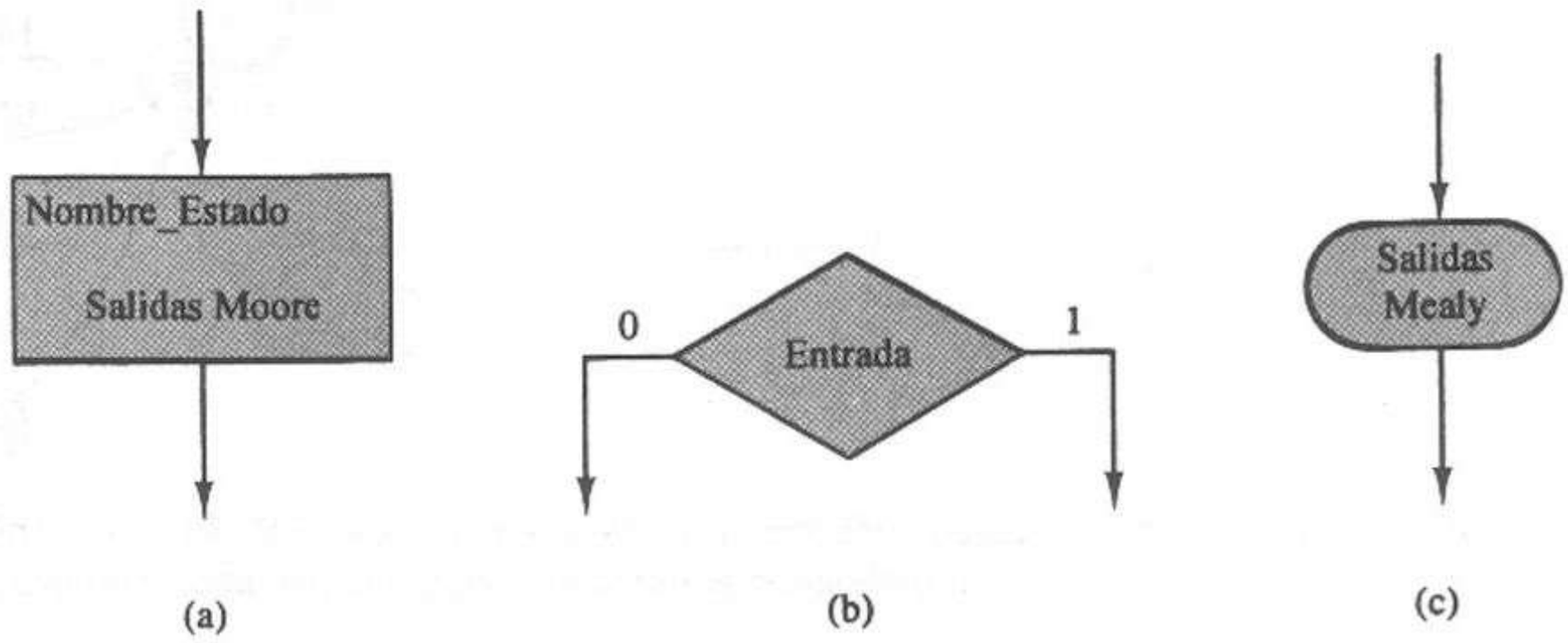


Figura 8.42 Elementos de un diagrama ASM. **(a)** Caja de estado. **(b)** Caja de decisión. **(c)** Caja de salida condicional.

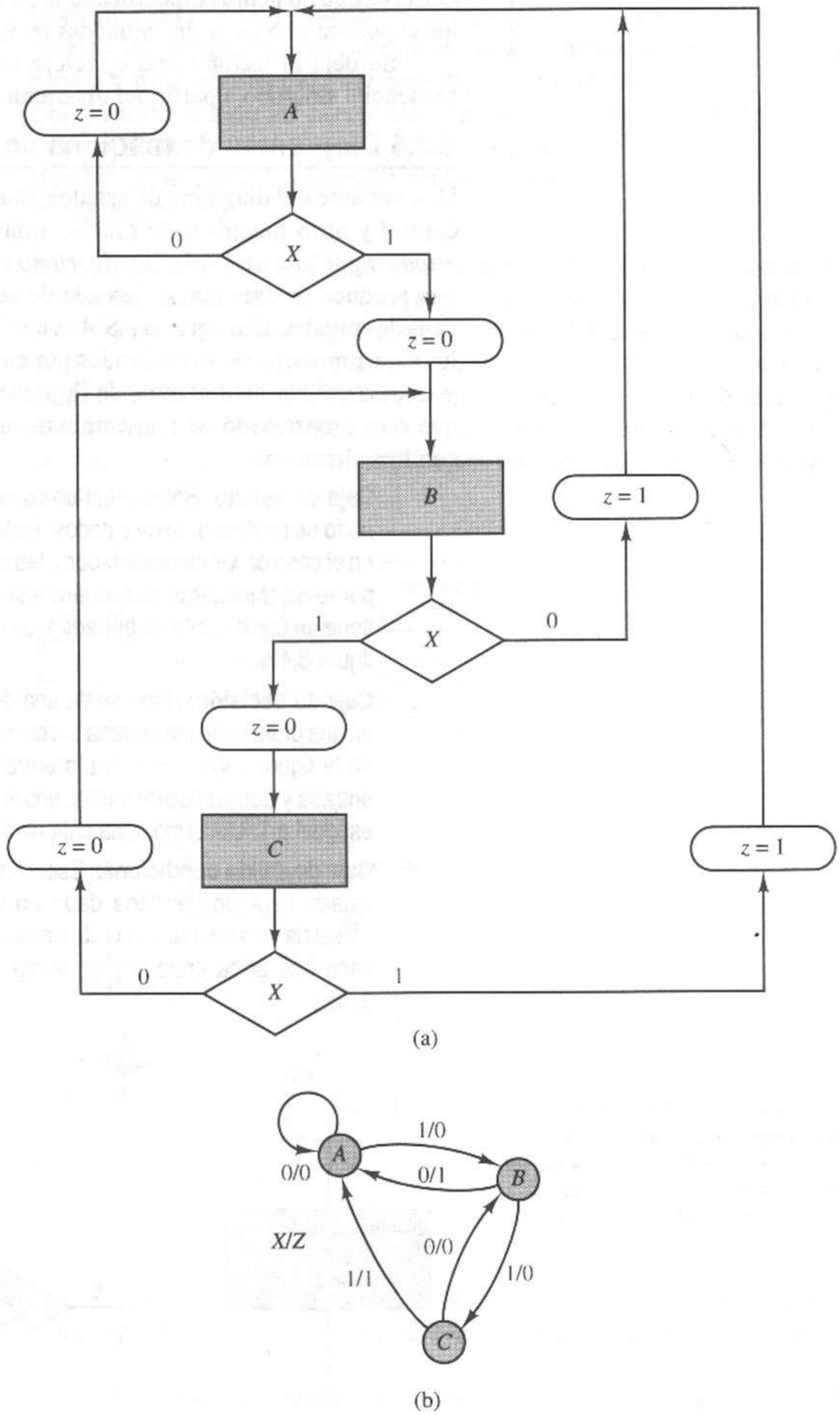


Figura 8.43 Representación ASM de un circuito secuencial en modelo de Mealy. **(a)** Diagrama ASM. **(b)** Diagrama de estados equivalente.

Podemos utilizar los diagramas ASM para desarrollar los circuitos con modelos Mealy y Moore. En las figuras 8.43a y b presentamos el diagrama ASM y el diagrama de estados correspondiente, respectivamente. En el diagrama ASM, observe que hemos especificado la salida z en cajas de salida condicional, una para cada estado y combinación de entradas. Esto corresponde a asociar la salida con los arcos del diagrama de estados.

El diagrama ASM y el diagrama de estados correspondiente para un circuito Moore aparecen en las figuras 8.44a y b, respectivamente. Observe que no hay cajas de salida condicional en el diagrama ASM. Las salidas de un circuito Moore sólo dependen de las variables de estado y, por tanto, se especifican dentro de las cajas de estado en el diagrama ASM y dentro de los nodos del diagrama de estados.

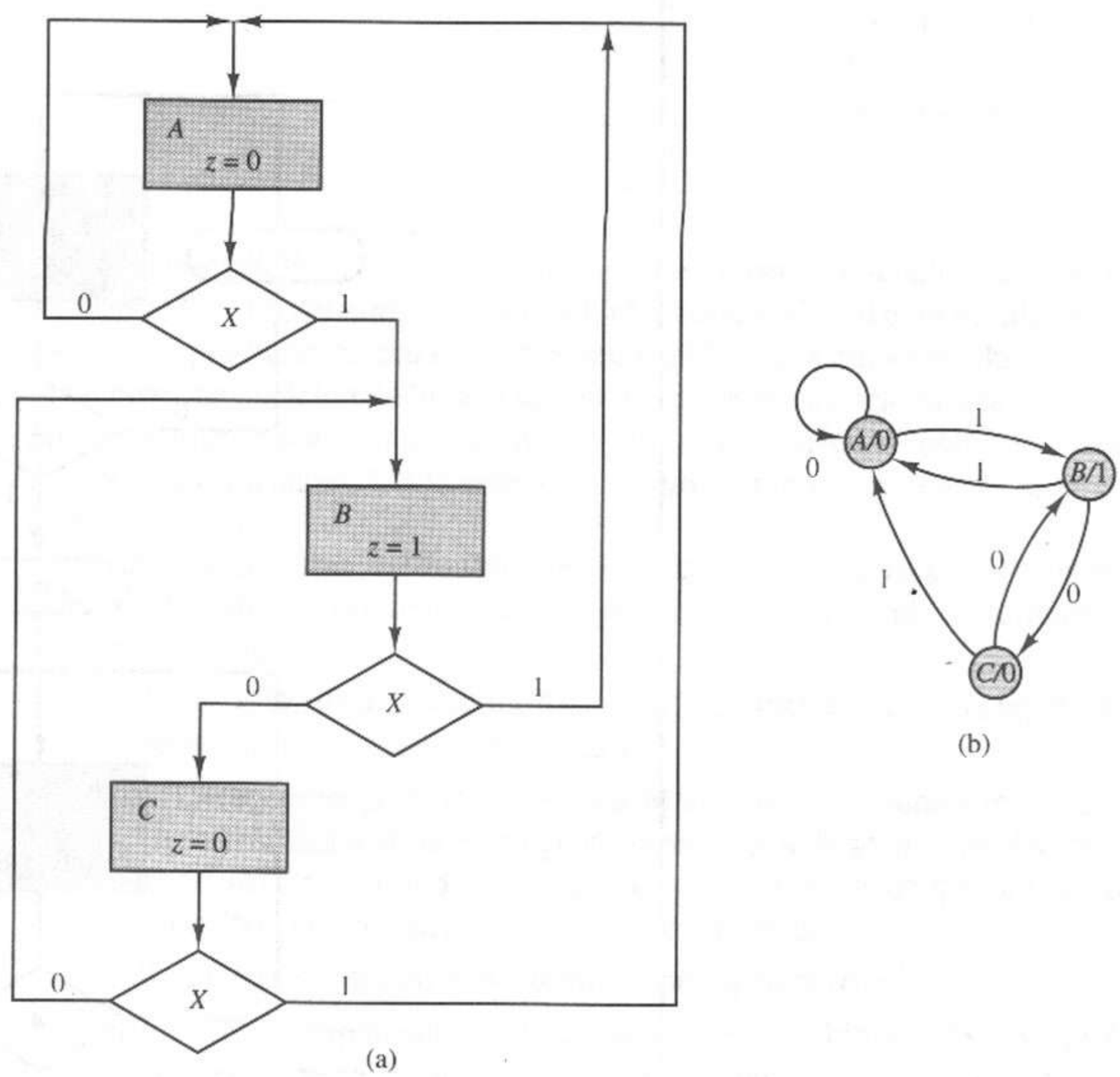


Figura 8.44 Representación ASM de un circuito secuencial en modelo de Moore. **(a)** Diagrama ASM. **(b)** Diagrama de estados equivalente.

Mealy

Mostramos el uso de los diagramas ASM en el diseño de circuitos secuenciales en los dos ejemplos siguientes.

EJEMPLO 8.19

Diseñar un diagrama ASM para un complementador a dos, serial, de 8 bits, que implante el algoritmo 1.4 presentado en el capítulo 1.

El algoritmo 1.4 nos pide examinar los bits del número, de derecha a izquierda, copiando los bits hasta copiar el primer bit 1; de ahí en adelante, hay que complementar los bits restantes. El diagrama ASM del algoritmo de complemento a dos serial aparece en la figura 8.45. Los bits de datos entran en el circuito secuencial por la entrada x, comenzando con el bit menos significativo. La salida del circuito z es el bit de datos corregido. En el estado A estamos buscando el primer bit 1 y, por tanto, $z = x$. En el estado B hemos detectado el primer bit 1, por lo que $z = \overline{x}$. Observe que hemos utilizado un modelo de Mealy, ya que z es una función de x y del estado. Por ello, especificamos todas las salidas en cajas de salida condicional.

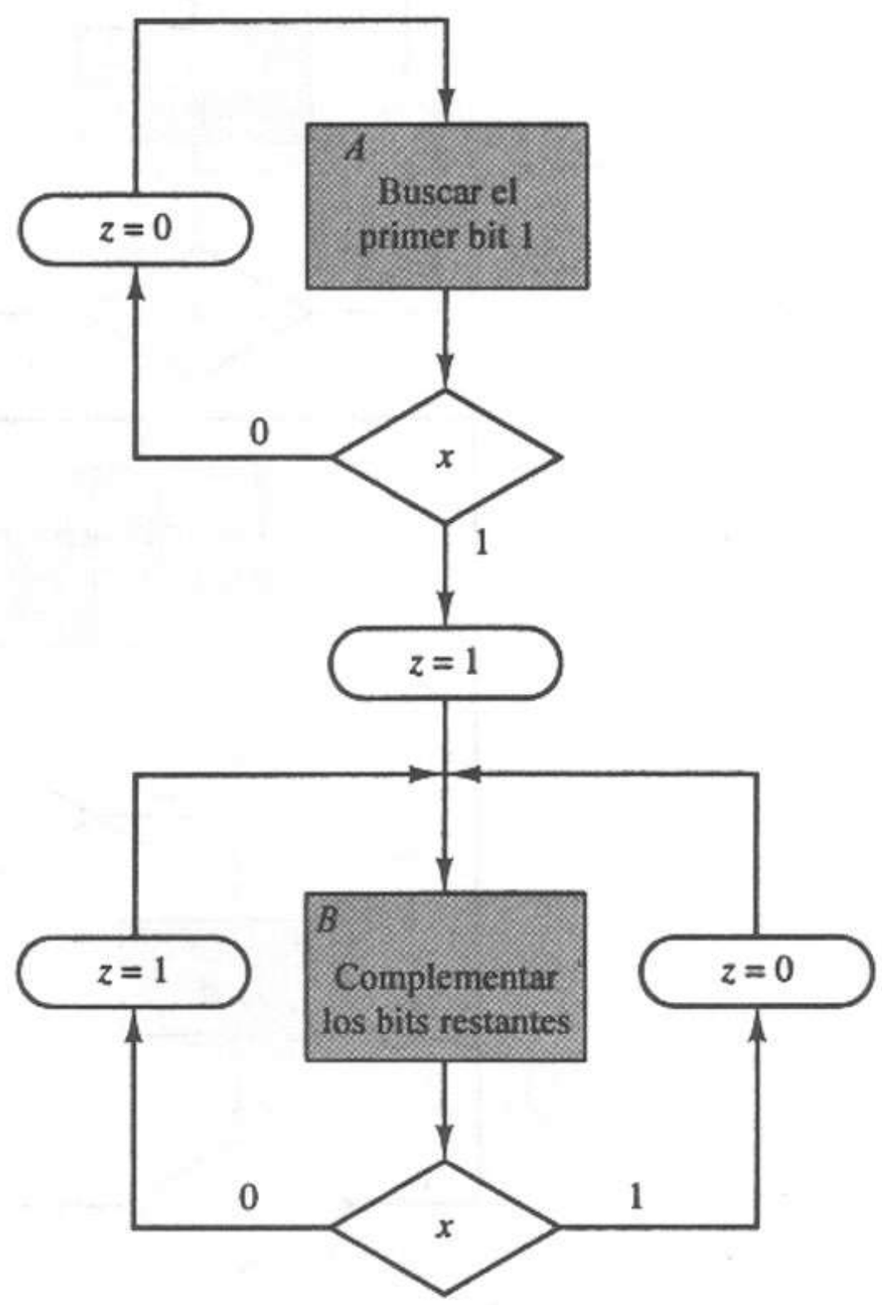


Figura 8.45 Representación ASM de un complementador a dos, serial.

Ahora veremos un problema de diseño más interesante, que implica una ruta de datos y una unidad de control.

EJEMPLO 8.20

Diseñar la unidad de control para un multiplicador binario que calcule el producto (8 bits) de dos números binarios sin signo de 4 bits mediante una serie de operaciones de suma y corrimiento. El multiplicador se inicia al pulsar una señal *Reset* y se detiene con el producto en su salida. Una señal *Alto* indica el fin de la operación.

Desarrollamos el algoritmo de multiplicación examinando primero el algoritmo de "lápiz y papel", como describimos en el capítulo 1. Considere el producto de $(0111)_2$ y $(1010)_2$.

				0	1	1	1	Multiplicando
×				1	0	1	0	Multiplicador
				0	0	0	0	Producto parcial 1
			0	1	1	1		Producto parcial 2
		0	0	0	0			Producto parcial 3
	0	1	1	1				Producto parcial 4
1	0	0	0	1	1	0		Producto

Examinamos los bits del multiplicador en forma secuencial, de derecha a izquierda. Si el bit del multiplicador es 1, el producto parcial es el multiplicando, y si es 0, el producto parcial es simplemente 0000. Cada nuevo producto parcial se recorre una posición de bit a la izquierda, antes de sumarlo al total. También podemos fijar la posición del producto parcial y recorrer el total a la derecha después de cada suma. Utilizamos este último método para nuestra unidad de control.

La ruta de datos del multiplicador binario requiere tres registros y un sumador binario, como vemos en la figura 8.46a. Los registros cumplen las siguientes funciones.

A: Un registro de corrimiento de 5 bits que retiene los cuatro bits más significativos del producto y el acarreo del sumador.

Q: Un registro de corrimiento de 4 bits. Al principio, Q contiene el multiplicador. En cada iteración, recorremos Q una posición a la derecha, reemplazando su bit del extremo izquierdo por un bit del producto, de modo que al final de la operación Q contendrá los 4 bits inferiores del producto.

M. Un registro paralelo de 4 bits que retiene el multiplicando.

Además de estos componentes, utilizamos un contador binario de 2 bits, *CNT*, para contar el número de iteraciones. Lo iniciamos con el valor 00 y lo incrementamos después de cada operación de corrimiento, regresando a 00 después de la cuarta iteración. Utilizamos una compuerta lógica para indicar la condición 00.

Calcularemos el producto sumando el multiplicando al total actual que está en el registro A, si el bit del multiplicador verificado es 1. En vez de sumar el producto parcial 0000 al total si el bit del multiplicador es 0, simplemente omitimos el paso de la suma.

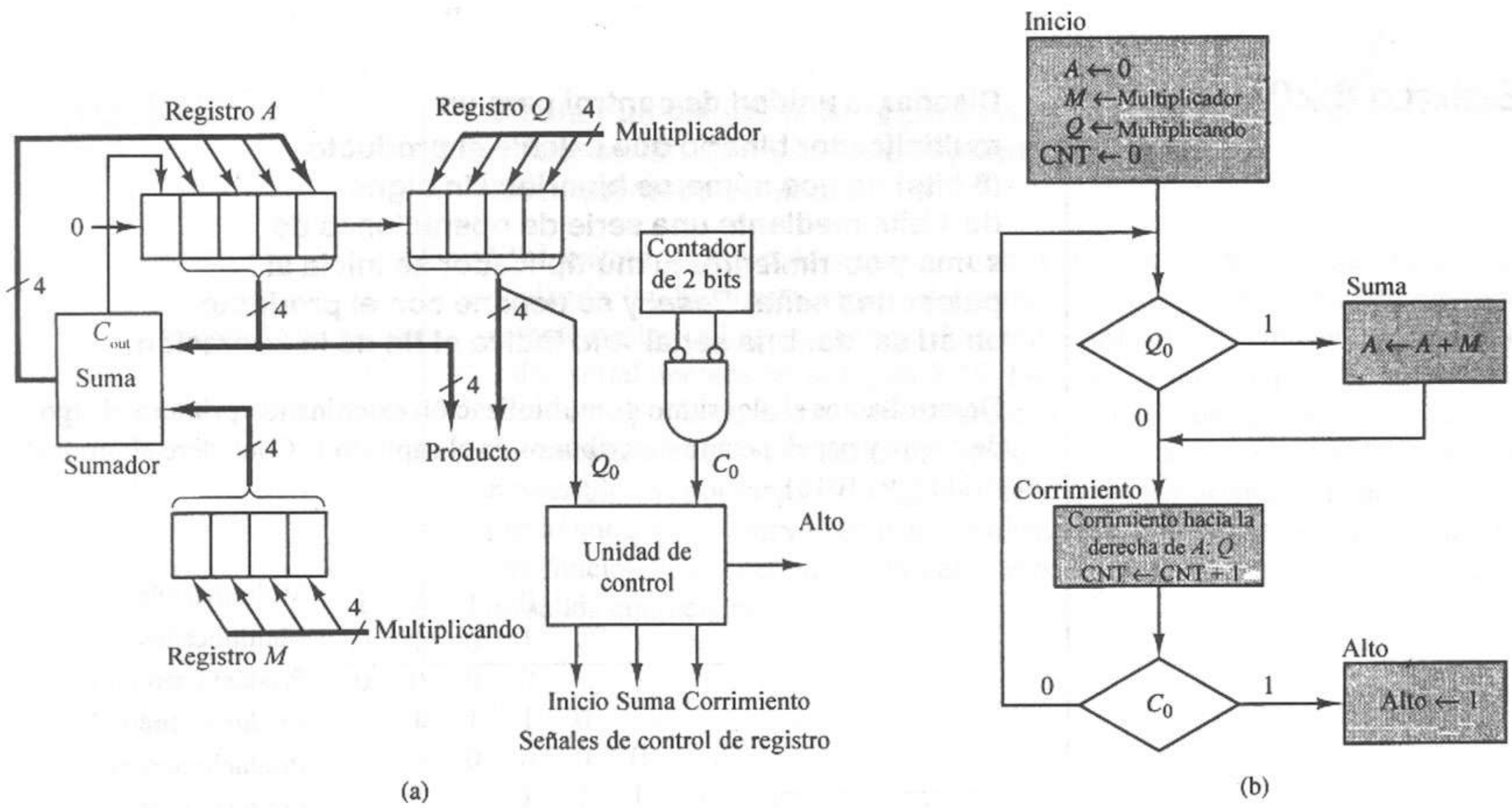


Figura 8.46 Multiplicador binario. **(a)** Diagrama de bloques. **(b)** Diagrama ASM.

El controlador tiene dos entradas: Q_0 es el bit del multiplicador que se está verificando y C_0 es 1 cuando el contador contiene 00, indicando el fin del cuarto ciclo de suma y corrimiento. El diagrama ASM del controlador aparece en la figura 8.46b. Utilizamos un modelo de Moore con los cuatro estados siguientes:

Inicio: Carga los operandos en los registros M y Q; limpia el registro A y el contador *CNT*.

Suma: Suma el multiplicando (M) al producto parcial actual (A) cargando las salidas de la suma y el acarreo del sumador binario en el registro A.

Corrimiento: Recorre el producto parcial y el multiplicador un bit a la derecha activando las líneas de control de corrimiento de los registros A y Q. Además, incrementa el contador activando su línea de control de incremento.

Alto: Detiene el proceso, dejando el producto final en los registros A y Q.

Verificamos un bit del multiplicador en cada iteración del algoritmo, para determinar si entra en el estado *Suma* o bien si se omite el estado *Suma* y se pasa directamente al estado *Corrimiento*. Se verifica la salida del contador después del estado *Corrimiento* para determinar si se debe parar o se realiza otra iteración.

El controlador tiene cuatro salidas, una correspondiente a cada estado. La salida *Inicio* activa las líneas de control de carga de los registros M y Q y las líneas para limpiar del registro A y del contador. La salida *Suma* activa la línea de carga del registro A, la salida *Corrimiento* activa las líneas de control de

corrimiento de los registros A y Q y la línea de control de incremento del contador. La salida *Alto* indica el fin de la operación.

8.3.6 Método de diseño de una máquina de estados finitos *one-hot*

En el diseño de circuitos secuenciales descritos mediante diagramas ASM empleamos con frecuencia un método de diseño simplificado que utiliza el método de asignación de estados *one-hot* para reducir el tiempo de diseño. Una *asignación de estados one-hot* utiliza una variable de estado (y en consecuencia un *flip-flop*) para cada estado, como se muestra en la tabla 8.1. Todas las variables de estado, excepto una, son iguales a 0 en un instante dado. La única variable de estado igual a 1 es el estado *hot*. Por tanto, un circuito secuencial de n estados necesita n variables de estado en vez de $\lceil \log_2 n \rceil$. El circuito resultante tiene más *flip-flops*, pero es más fácil de diseñar y con frecuencia requiere menos compuertas lógicas combinatorias.

Podemos deducir los diseños *one-hot* en forma directa de los diagramas ASM. La figura 8.47 muestra algunas implantaciones de circuitos lógicos a partir de las diversas construcciones ASM. Utilizamos *flip-flop* D para todos los elementos de memoria, con un *flip-flop* por cada caja de estado. Implantamos una seriación de estados simple al conectar en cascada *flip-flops*, como se ilustra en la figura 8.47a. En el estado A, la salida del *flip-flop* cumple $Q_A = 1$. Como $D_B = Q_A$, la salida del *flip-flop* Q_B será 1 en el siguiente pulso de reloj, mientras que Q_A regresa a 0. De esta manera, el estado *hot* pasa de un *flip-flop* al siguiente. Para los circuitos Moore, las salidas del circuito son simplemente las salidas del *flip-flop*, como se muestra en la figura 8.47a, ya que sólo dependen del estado.

Las rutas de control que se unen en el diagrama ASM se unen mediante una compuerta OR en el circuito de control, como muestra la figura 8.47b. En este circuito, la salida del *flip-flop* Q_B será 1 en el siguiente pulso de reloj si $Q_A = 1$ o $Q_C = 1$.

Implantamos las cajas de decisión ASM como en la figura 8.47c. En este caso, cuando la salida del *flip-flop* $Q_A = 1$ y la entrada $x = 0$, las compuertas AND establecen las entradas del *flip-flop* como $D_B = 1$ y $D_C = 0$, haciendo que la salida del *flip-flop* Q_B asuma el valor 1 en el siguiente pulso de reloj. Si $x = 1$, entonces $D_B = 0$ y $D_C = 1$.

Realizamos las cajas de salida condicional de un diagrama ASM conectando las salidas del circuito con las salidas de las compuertas AND que implantan las cajas de decisión, como se muestra en la figura 8.47c, ya que estas señales dependen del estado y de la entrada.

TABLA 8.1 ASIGNACIÓN DE ESTADOS PARA UN CIRCUITO SECUENCIAL DE CUATRO ESTADOS

Estado	Asignación secuencial y_1y_0	Asignación *one-hot* $y_3y_2y_1y_0$
A	00	0001
B	01	0010
C	10	0100
D	11	1000

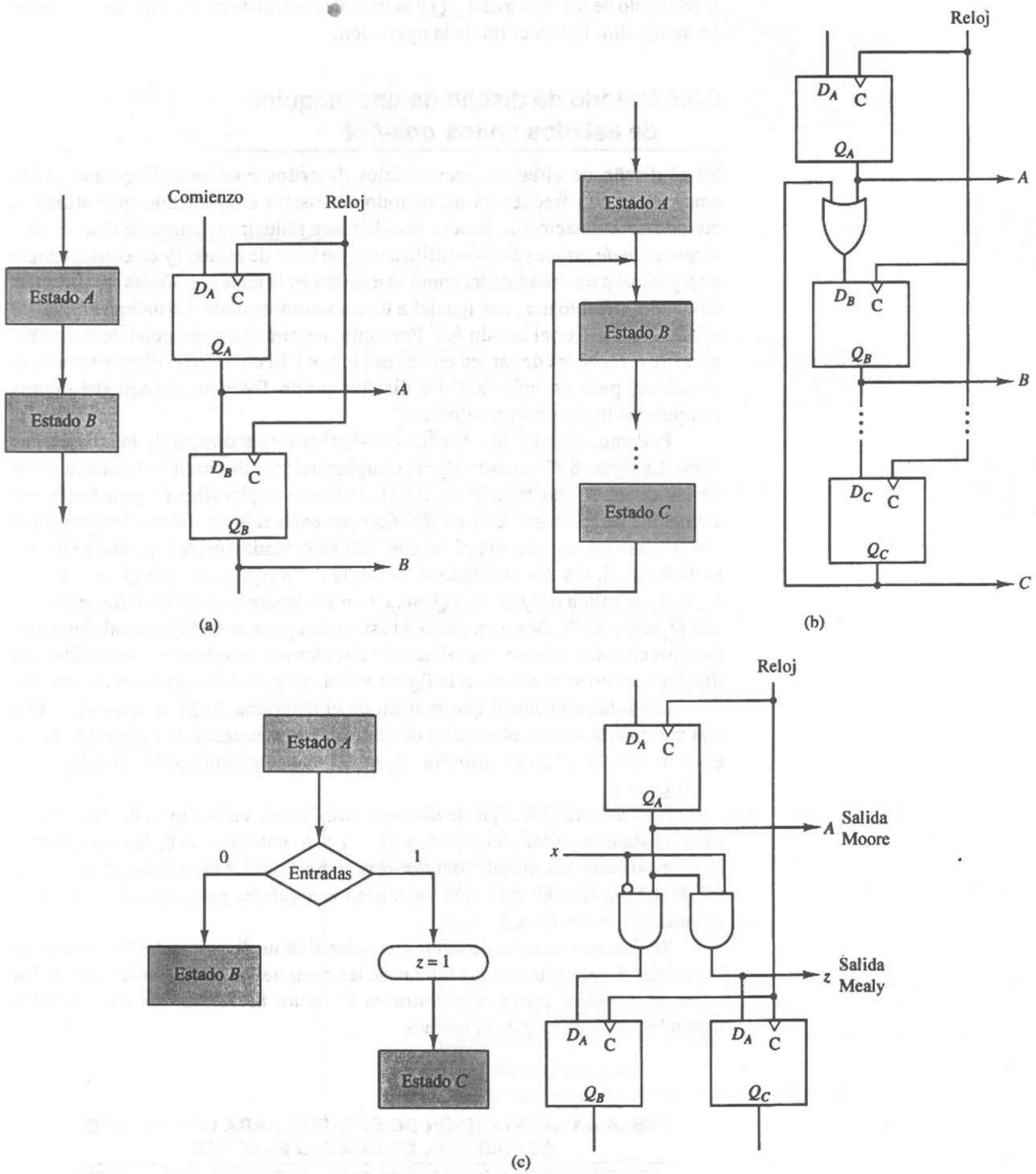


Figura 8.47 Controladores diseñados con el método *one-hot*. **(a)** Secuenciación de estados simple. **(b)** Fusión de rutas de control. **(c)** Decisión de control.

El siguiente ejemplo ilustra el método de diseño *one-hot* para deducir los circuitos de los modelos Moore y Mealy a partir de los diagramas ASM.

EJEMPLO 8.21

Utilizar el método de diseño *one-hot* para implantar el diagrama ASM de la unidad de control del multiplicador de la figura 8.46b y el diagrama ASM del complementador a dos serial de la figura 8.45.

La figura 8.48a muestra el diagrama lógico de la unidad de control del multiplicador. Observe que la estructura del circuito se deduce directamente del diagrama ASM. Como éste es un modelo de Moore, las salidas sólo son controladas por las cuatro salidas de *flip-flop*. Iniciamos el algoritmo proporcionando un pulso en la línea de control *Comienzo* para que el primer *flip-flop* asuma el valor 1 en un ciclo de reloj y después pase a 0 en el siguiente ciclo de reloj.

La figura 8.48b muestra el circuito complementador a dos serial. De nuevo, la estructura del circuito se deduce directamente del diagrama ASM. En este caso utilizamos un modelo de Mealy. Por tanto, la salida z es función de las variables de estado y de la entrada x. En este caso, $z = 1$ si el circuito está en el estado 0 y $x = 1$ o bien si el circuito se encuentra en el estado 1 y $x = 0$.

8.4 Circuitos con especificación incompleta

Un circuito secuencial tiene una *especificación incompleta* si su tabla de estados contiene condiciones prescindibles. Esto surge por lo general en circuitos en los que sólo se puede aplicar cierto conjunto de entradas; por tanto, los estados y salidas que podrían presentarse como entradas prohibidas nunca ocurren y podemos asignarlas como prescindibles. El siguiente ejemplo ilustra esta idea.

EJEMPLO 8.22

Diseñar un circuito detonador como el que se muestra en la figura 8.49a, que exhibe el comportamiento del diagrama de estados de la figura 8.49b.

Cuando el dispositivo está activo y $x = 0$, permanece en un estado inactivo A. La secuencia de detonación se inicia haciendo $x = 1$. El dispositivo pasa del estado A al B, después C y por último D, donde emite un pulso ($z = 1$) para detonar un explosivo. Todo el circuito antes del detonador está diseñado de modo que una vez que se presenta el primer $x = 1$, el dispositivo no puede regresar a su estado anterior; es decir, no ocurre $x = 0$ una vez que se tiene $x = 1$.

Las figuras 8.49b y c muestran el diagrama de estados parcial y la tabla de estados completa para el detonador, respectivamente. Observe de nuevo que, una vez iniciada la secuencia del detonador, ésta continúa sin interrupción hasta generar el pulso de detonación. El estado final es prescindible, pues el explosivo se ha encendido. En el siguiente análisis realizaremos el circuito detonador mediante *flip-flops* T con reloj. Si elegimos la asignación de estados (y_2y_1) como

$$A = 00 \qquad C = 10$$
$$B = 01 \qquad D = 11$$

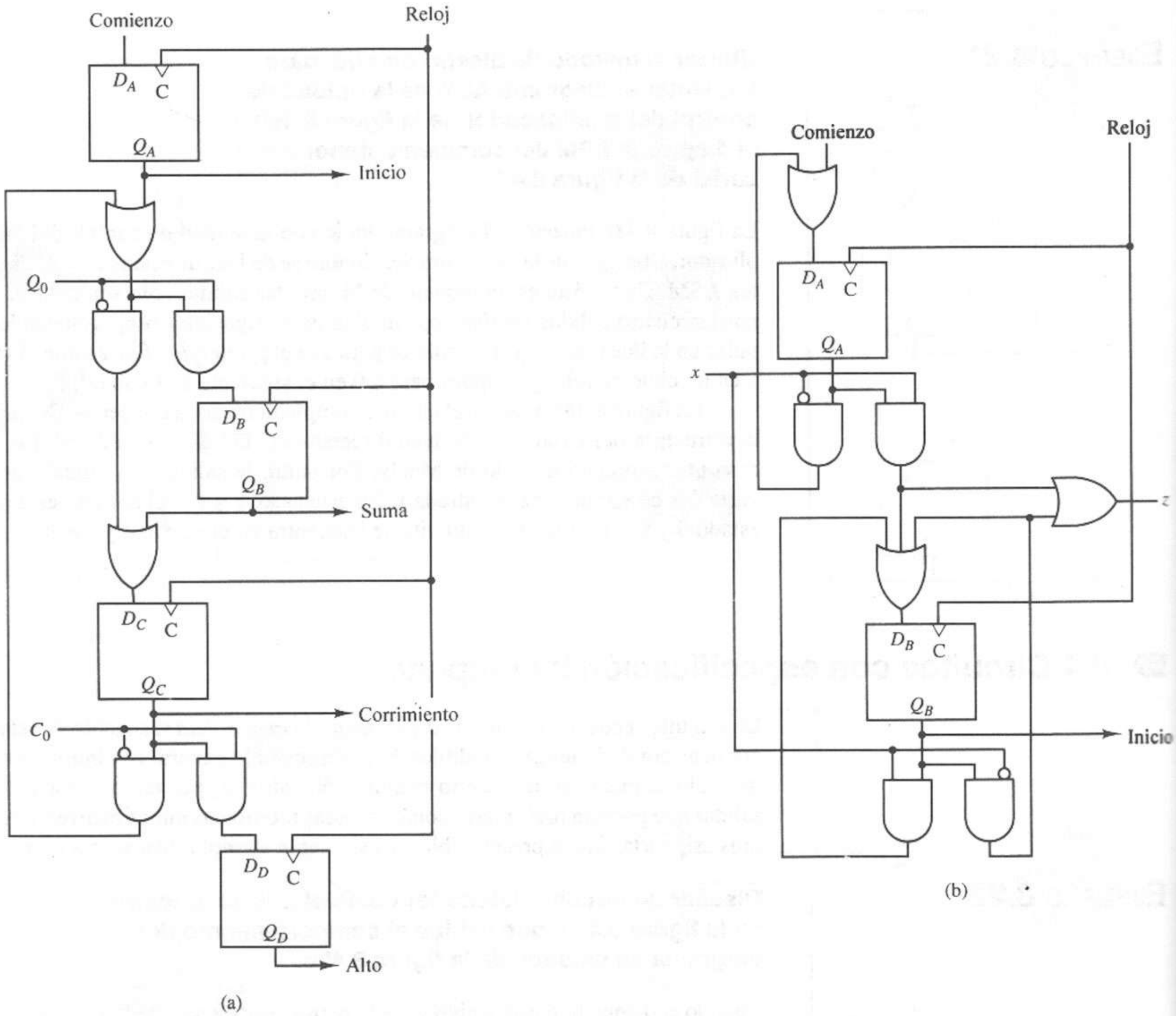


Figura 8.48 Controladores diseñados con el método *one-hot*. **(a)** Unidad de control del multiplicador. **(b)** Unidad de control del complementador serial a dos.

la figura 8.50 muestra todas las tablas necesarias para la realización del circuito. Las siguientes ecuaciones son una consecuencia directa de las tablas:

$$T_1 = x$$
$$T_2 = y_1$$
$$z = y_1 y_2$$

La figura 8.51 muestra el circuito real para el detonador. Como nuestro análisis sólo es válido durante el pulso de reloj, utilizamos el pulso de reloj para enviar la salida por una compuerta.

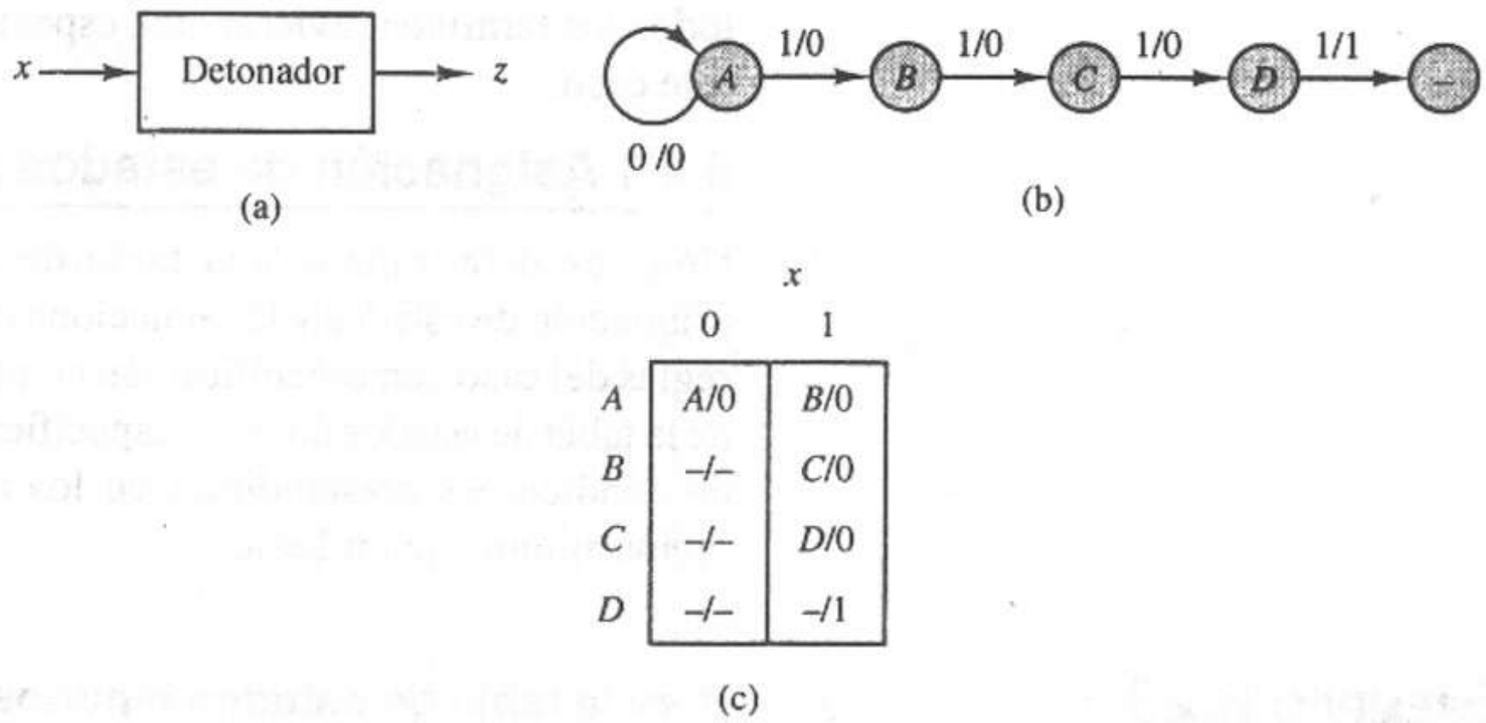


Figura 8.49 Circuito detonador. **(a)** Diagrama de bloques. **(b)** Diagrama parcial de estados. **(c)** Tabla de estados.

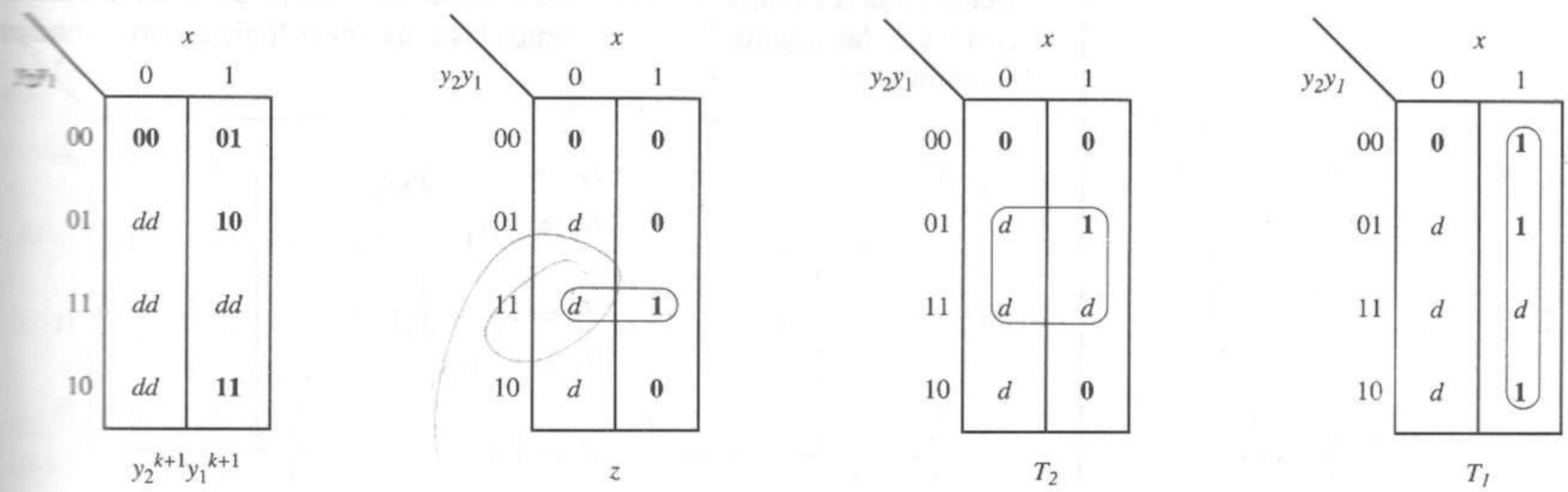


Figura 8.50 Mapas de transición, de salida y de excitación del detonador.

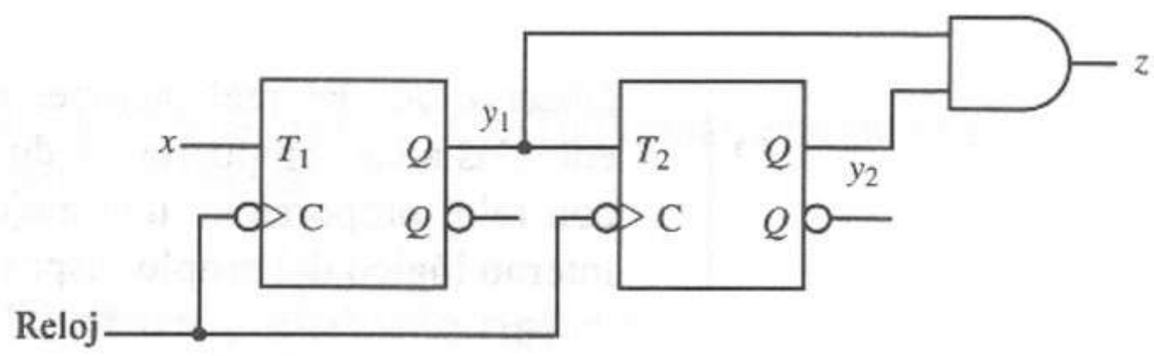


Figura 8.51 Implantación del detonador.

El lector debe observar que los circuitos con especificación incompleta tienen una ventaja sobre los circuitos con especificación completa desde el punto de vista de la realización del hardware. La ventaja surge de la presencia de condiciones prescindibles en la tabla de estados. En otras palabras, podemos agrupar estas condiciones prescindibles con las condiciones de los mapas de excitación a fin de obtener un circuito más sencillo que el que tendríamos si

todos los términos tuvieran una especificación completa. La figura 8.50 ilustra este caso.

8.4.1 Asignación de estados y realización de circuitos

Una vez determinada una tabla de estados reducida, podemos obtener la asignación de estados y las ecuaciones lógicas del circuito utilizando las mismas reglas del caso con especificación completa. Sin embargo, como ciertas entradas de la tabla de estados no están especificadas, existe por lo general un gran número de condiciones prescindibles en los mapas K generados, lo que implica una mejor minimización lógica.

EJEMPLO 8.23

Para la tabla de estados binarios de la figura 8.52a, completar la realización mediante *flip-flops* D, T con reloj y JK con reloj.

Deducimos los mapas K para cada realización a partir de la tabla binaria de estados de las figuras 8.52b a e; damos las ecuaciones lógicas correspondientes a continuación:

$$D_2 = \bar{x}y_1 + y_2y_1$$
$$D_1 = y_2\bar{y}_1$$

$$T_2 = \bar{x}y_1 + y_2\bar{y}_1$$
$$T_1 = y_2 + y_1$$

$$J_2 = \bar{x}y_1$$
$$K_2 = \bar{y}_1$$
$$J_1 = y_2$$
$$K_1 = 1$$

$$z = xy_2 + \bar{x}\bar{y}_1$$

Observe que las realizaciones con *flip-flops* D y T con reloj requieren ocho entradas a las compuertas, y dos la de JK con reloj. En general, el *flip-flop* JK con reloj proporciona una mejor reducción lógica, ya que tiene más control interno lógico del propio dispositivo.

8.5 Diseño asistido por computadora de circuitos secuenciales

El proceso de diseño, verificación, construcción y prueba de circuitos lógicos secuenciales es mucho más complejo que en el caso de los circuitos lógicos combinatorios. Esta complejidad aumenta de manera significativa con el número de estados del circuito. En consecuencia, los métodos de diseño asistido por computadora (CAD) son, de hecho, una necesidad para el desarrollo de cualquier circuito secuencial con más de unos cuantos estados. En esta sección examina-

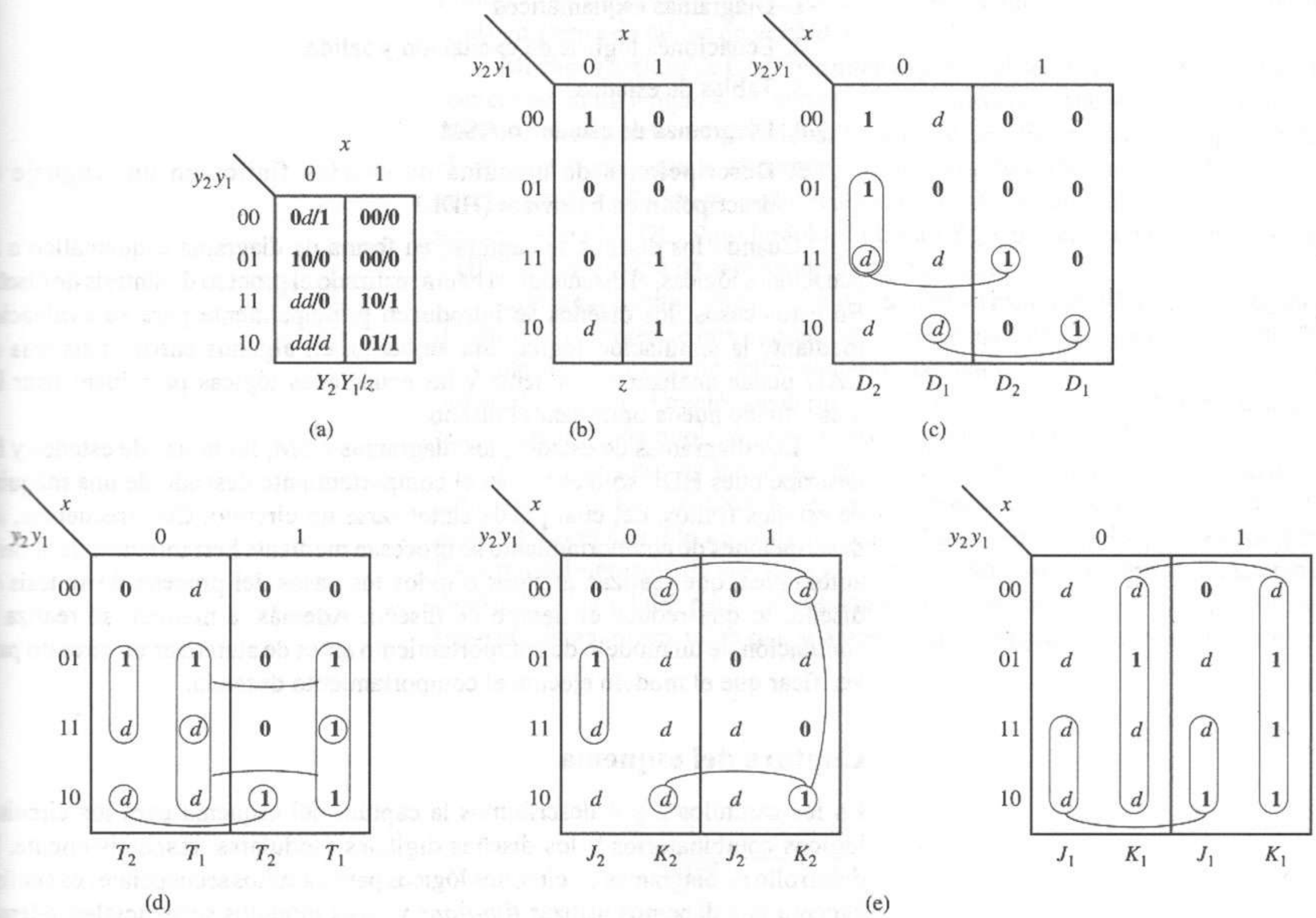


Figura 8.52 Realizaciones lógicas. **(a)** Tabla de estados binarios. **(b)** Mapa K para *z*. **(c)** *Flip-flops* D. **(d)** *Flip-flops* T con reloj. **(e)** *Flip-flops* JK con reloj.

remos algunas herramientas de CAD utilizadas en la síntesis y análisis de circuitos secuenciales síncronos.

8.5.1 Captura y síntesis del diseño

Iniciamos el diseño de un circuito secuencial expresando el comportamiento deseado del circuito en forma de diagrama de estados, diagrama ASM, tabla de estados o alguna otra descripción formal. A continuación procesamos esta descripción para sintetizar un circuito lógico que ejecute el comportamiento deseado. Existen muchas herramientas de CAD para procesar diversas representaciones de los circuitos secuenciales. Algunas de estas herramientas realizan en forma automática uno o más pasos del proceso de síntesis. En diferentes etapas del proceso de diseño, podemos utilizar las herramientas de simulación lógica para simular la operación del circuito y facilitar la verificación del diseño lógico o analizar su comportamiento respecto del tiempo.

Muchos sistemas CAD apoyan la creación y la captura de descripciones de circuitos secuenciales en varios formatos, que pueden incluir los siguientes:

1. Diagramas esquemáticos
2. Ecuaciones lógicas de excitación y salida
3. Tablas de estados
4. Diagramas de estados o ASM
5. Descripciones de máquina de estados finitos en un lenguaje de descripción de hardware (HDL)

Cuando los diseños se capturan en forma de diagrama esquemático o de ecuaciones lógicas, el diseñador ya habrá realizado el proceso de síntesis de diseño. En estos casos, los diseños se introducen principalmente para su evaluación mediante la simulación lógica. Sin embargo, en algunos casos el sistema de CAD puede analizar el esquema y las ecuaciones lógicas para identificar las áreas donde pueda optimizar el diseño.

Los diagramas de estados, los diagramas ASM, las tablas de estados y las descripciones HDL sólo expresan el comportamiento deseado de una máquina de estados finitos, del cual puede sintetizarse un circuito. Con frecuencia, las descripciones de comportamiento se procesan mediante herramientas de síntesis automática, que realizan algunos o todos los pasos del proceso de síntesis de diseño, lo que reduce el tiempo de diseño. Además, a menudo se realiza la simulación de un modelo de comportamiento antes de sintetizar un circuito para verificar que el modelo ejecuta el comportamiento deseado.

Captura del esquema

En los capítulos 2 y 4 describimos la captura del esquema para los circuitos lógicos combinatorios y los diseños digitales modulares, respectivamente. El desarrollo de diagramas de circuitos lógicos para circuitos secuenciales es similar, excepto que debemos utilizar *flip-flops* y otros módulos secuenciales, además de las compuertas lógicas básicas y, por tanto, se incluyen en una o más bibliotecas de componentes. Además, hay que definir señales de reloj para el control de las transiciones de estado y especificar las señales *preset* y *clear* asíncronas para inicializar los estados de los *flip-flops*.

También hay que especificar una serie de parámetros de dispositivos en los circuitos secuenciales que no se aplican a los circuitos combinatorios. Los parámetros usuales son los tiempos de activación y de retención para los *latches*, los *flip-flops* y las entradas de otros módulos secuenciales; las especificaciones de reloj y los retardos por propagación de entradas síncronas y asíncronas a las salidas. Con frecuencia, estos parámetros se especifican para cada componente en una biblioteca de diseño, a fin de reflejar las características reales de la tecnología de ese componente. Además de las características físicas, algunos sistemas de diseño permiten asignar estados iniciales por omisión a elementos de memoria y emplearlos durante la simulación lógica.

Descripciones de la máquina de estados finitos

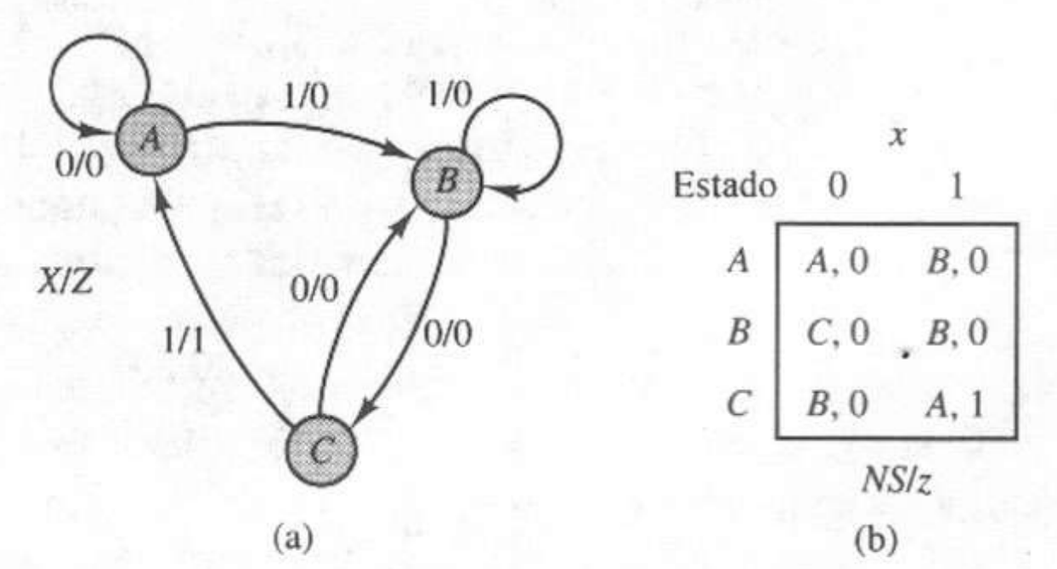
Los diagramas de estados, tablas de estados y otras descripciones de máquinas de estados finitos contienen básicamente la misma información: describen el comportamiento deseado del circuito en términos de transiciones de estado y cambios de salida en respuesta a todas las combinaciones posibles de entradas y estados.

Por ejemplo, las figuras 8.53a y b presentan un diagrama de estados y la correspondiente tabla de estados, respectivamente, de una máquina de estados finitos. Con frecuencia, las tablas de estados se introducen en un sistema de CAD en forma de tablas de verdad, como en la figura 8.53c.

Muchos sistemas de CAD permiten expresar el comportamiento del circuito secuencial en un lenguaje de descripción de hardware estándar, como VHDL o Verilog; otros pueden proporcionar un HDL específico para este propósito. En el capítulo 2 analizamos el uso de VHDL para describir los circuitos lógicos combinatorios. También podemos expresar fácilmente el comportamiento secuencial en VHDL. Por ejemplo, la figura 8.54 presenta una descripción en VHDL del diagrama de estados de la figura 8.53a.

Como explicamos en el capítulo 2, una descripción en VHDL de un módulo de circuito comprende una entidad, que define las entradas y salidas del módulo, y una arquitectura, que define el comportamiento o la implantación estructural del módulo. La declaración de entidad de la figura 8.54 define el circuito *seqckt* con una entrada de reloj, *clk*, una entrada de señal, *x* y una salida *z*.

Por lo general, en una descripción de arquitectura VHDL se describe el comportamiento de un circuito secuencial mediante una o más estructuras de *proceso*. En el ejemplo de la figura 8.54, el proceso *clock* describe el comportamiento del elemento de memoria y el proceso *state_trans* define las transiciones de estado y las salidas para cada estado y combinación de entradas. Éste último tiene la misma información que aparecería en un diagrama de estados o tabla de



Estado actual	Entradas x	::	Estado siguiente	Salidas z
A	0	::	A	0
A	1	::	B	0
B	0	::	C	0
B	1	::	B	0
C	0	::	B	0
C	1	::	A	1

(c)

Figura 8.53 Descripción del circuito secuencial. **(a)** Diagrama de estados. **(b)** Tabla de estados. **(c)** Formato de tabla de verdad.

```
ENTITY seqckt IS
  PORT(clk: in  bit;    -- señal de reloj
       x:   in  bit;    -- entrada del circuito
       z:   out bit);   -- salida del circuito
END seqckt;


ARCHITECTURE mealy OF seqckt IS
   TYPE states IS (a,b,c);              -- tres estados
   SIGNAL state:      states := a; -- estado actual (inicialmente a)
   SIGNAL next_state: states := a; -- estado siguiente (valor inicial a)
BEGIN
clock: PROCESS(clk)       -- reacciona a la transición sobre la señal clk
        BEGIN
          IF clk'EVENT and clk = '1' THEN    -- transición ascendente de clk
            state <= next_state;             -- cambio de estado
          END IF;
        END PROCESS clock;
state_trans:  PROCESS(state,x) -- reacciona a cambios en el estado o en la entrada x
                BEGIN
                  next_state <= state;  -- actualiza next_state
                  CASE state IS          -- transiciones de estado y salidas
                     WHEN a => IF x = '0' THEN
                                  z <= '0';
                               ELSE
                                  next_state <= b;
                                  z <= '0';
                               END IF;
                     WHEN b => IF x = '0' THEN
                                  next_state <= c;
                                  z <= '0';
                               ELSE
                                  z <= '0';
                               END IF;
                     WHEN c => IF x = '0' THEN
                                  next_state <= b;
                                  z <= '0';
                               ELSE
                                  next_state <= a;
                                  z <= '1';
                               END IF;
                  END CASE;
              END PROCESS state_trans;
END mealy;
```

Figura 8.54 Descripción en VHDL de un circuito secuencial.

estados. Observe que definimos un tipo especial de datos, *states*, con valores (*a*, *b*, *c*). Esto permite especificar el estado y el estado siguiente del circuito de manera simbólica hasta determinar la asignación de estados.

El enunciado *clock* PROCESS se ejecuta en cada transición de la señal *clk*. Dentro del proceso, la proposición IF permite cambiar a *state* sólo cuando *clk* hace una transición y *clk* = 1. Por tanto, los cambios de estado ocurren en la transición ascendente de *clk*.

El enunciado *state_trans* PROCESS reacciona a los cambios en *state* o en la entrada *x* y, por tanto, modela la parte lógica combinatoria del circuito secuencial. El enunciado CASE especifica un valor de estado siguiente y una salida para cada estado y valor de entrada y, por tanto, describe un modelo de Mealy.

El lector interesado debe consultar [10,11] para obtener mayor información acerca del uso de VHDL en el modelo de circuitos secuenciales.

Síntesis lógica automatizada por computadora

Con frecuencia se utilizan herramientas de síntesis automatizadas por computadora para realizar algunos o todos los pasos del proceso de diseño de un circuito secuencial, partiendo de una descripción del comportamiento del circuito en forma de diagrama de estados, tabla de estados o descripción HDL y, por último, generar un diagrama del circuito lógico o una lista de red.

En la mayor parte de los casos, el diseñador tiene varias opciones para dirigir la síntesis de un diseño. Estas opciones se especifican antes de iniciar el sintetizador y pueden incluir lo siguiente.

1. *Seleccionar un método de asignación de estados*. Algunas herramientas necesitan que el diseñador proporcione una asignación de estados específica para utilizarse durante la síntesis o bien que designe uno de varios patrones estándar para la asignación de estados. Por ejemplo, la tabla 8.2 enumera tres opciones de asignación de estados, proporcionadas por la herramienta de síntesis Mentor Graphics *AutoLogic* [11]. También existen otras opciones de asignación.

   Algunas de las herramientas de síntesis lógica más avanzadas incluyen algoritmos para deducir una asignación de estados óptima a partir de una tabla de estados dada. Describiremos algunos de estos algoritmos en el capítulo 9.

2. *Seleccionar los tipos de flip-flop*. En algunos programas de síntesis es necesario designar los tipos de *flip-flop* específicos. Otros eligen un tipo

TABLA 8.2 OPCIONES DE ASIGNACIÓN DE ESTADOS PARA UNA MÁQUINA DE CUATRO ESTADOS

Estado	Secuencial	Código Gray	One hot
A	00	00	0001
B	01	01	0010
C	10	11	0100
D	11	10	1000

por omisión o, en sistemas más avanzados, intentan determinar el tipo óptimo para el circuito dado. Con la automatización por computadora muchas veces es posible sintetizar un diseño de varias formas, utilizando diferentes opciones de *flip-flop* para determinar el que requiere un mínimo de compuertas lógicas combinatorias.

3. *Especificar algoritmos de minimización lógica combinatoria para las ecuaciones de excitación y de salida.* Estas ecuaciones se deducen de las tablas de excitación y de salida de los *flip-flops*. Podemos aplicar muchos de los métodos descritos en el capítulo 3 para minimizar estas ecuaciones. Con frecuencia, el diseñador dispone de varios algoritmos y debe decidir cuál de ellos utiliza equilibrando el grado de optimización y el tiempo de diseño.

La figura 8.55 muestra un circuito secuencial sintetizado en forma automática a partir de la descripción VHDL de la figura 8.54, mediante la herramienta de síntesis *AutoLogic VHDL* de Mentor Graphics [11]. Hemos producido este circuito en dos pasos. En el primero convertimos la descripción VHDL en ecuaciones lógicas y en un circuito lógico generado sin optimización. Indicamos al sintetizador que utilizara *flip-flops* D y la asignación secuencial de estados de la tabla 8.2. El circuito resultante tiene 17 compuertas lógicas primitivas y seis multiplexores, además de dos *flip-flops* D. Esta herramienta en particular sólo utiliza compuertas lógicas de dos entradas y multiplexores 2 a 1 para implantar la sección lógica combinatoria. El segundo paso optimizó el diseño al de la figura 8.55, reduciendo la parte lógica combinatoria a cuatro compuertas y un multiplexor.

El lector debe consultar [12] si desea mayor información acerca de los algoritmos y métodos de la síntesis lógica automática.

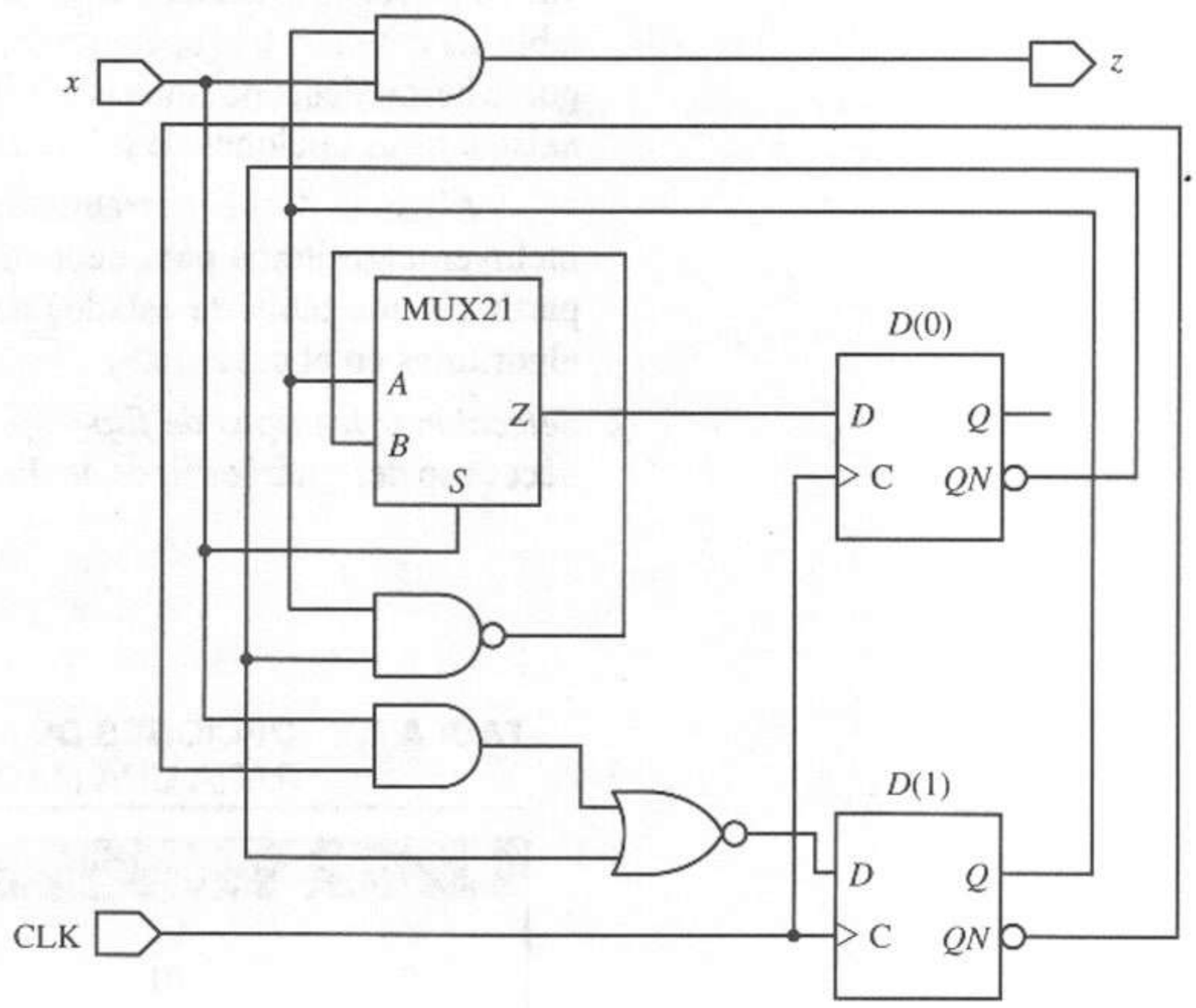


Figura 8.55 Circuito lógico sintetizado a partir de una descripción VHDL.

8.5.2 Análisis y verificación del diseño

Con mucha frecuencia, el análisis y la verificación de un diseño de circuito secuencial se realiza mediante una simulación lógica. La verificación del circuito secuencial puede incluir un análisis funcional para garantizar que el circuito realiza el comportamiento definido en la tabla de estados u otras descripciones de máquina de estados finitos, y un análisis de tiempo para determinar sus características de desempeño.

Análisis funcional

El análisis funcional de un diseño de circuito secuencial sirve para verificar que la función lógica realizada por el circuito concuerda con el comportamiento lógico descrito en el comportamiento inicial, es decir, verificar que las salidas son correctas y que ocurren las transiciones de estado adecuadas para cada combinación estado-entrada, definidas en la tabla de estados u otra descripción de máquina de estados finitos.

Por lo general, es recomendable simular la operación de una máquina de estados finitos a partir de su descripción de comportamiento antes de iniciar el proceso de síntesis para verificar que el algoritmo sea correcto. El resultado de esta simulación del comportamiento se puede comparar entonces con los resultados de la simulación del circuito lógico sintetizado para garantizar que el circuito lógico sí tiene el comportamiento deseado.

En el capítulo 2 describimos la simulación lógica de los circuitos lógicos combinatorios. Este proceso es similar para los circuitos secuenciales síncronos, aunque existen varias consideraciones adicionales.

1. Debemos definir una señal de reloj en términos de su periodo y ciclo de trabajo, es decir, la parte del periodo del reloj durante la cual el reloj es alto.
2. Hay que definir las formas de onda de las señales por aplicar a las entradas del circuito. Observe que es preciso coordinar el tiempo de cada cambio de entrada con los tiempos de transición del reloj de modo que todas las entradas de excitación de *flip-flop* sean estables antes de cada transición de reloj. Por ejemplo, si definimos un reloj con transiciones cada 100 ns, a partir del instante 0, entonces los cambios de entrada no deben ocurrir en ningún momento que sea múltiplo de 100 ns.

   Algunos simuladores lógicos permiten especificar las secuencias de valores de entrada en forma de lista. Así, los valores se aplican en forma automática a las entradas antes de cada transición de reloj.
3. Hay que especificar el estado inicial del circuito; es decir, todas las salidas de los elementos de memoria deben tener valores iniciales específicos. Algunos simuladores permiten asignar estados por omisión como valores iniciales. Otros asignan a cada *flip-flop* un valor de estado desconocido especial, por lo general, X o U, para indicar que el *flip-flop* aún no asume un valor durante el funcionamiento del circuito. La persistencia de un valor X en la salida de un *flip-flop* indica que éste aún no tiene un valor específico 0 o 1, lo que puede indicar un error en el diseño.

   Por ejemplo, si un *flip-flop* T asume inicialmente un estado desconocido X, la alternancia del *flip-flop* no basta para determinar el estado siguiente del mismo, y entonces el estado del *flip-flop* mantendrá el valor X durante la simulación. Esto advierte al diseñador que debe proporcionar una señal *reset* o algún otro medio para inicializar el estado del *flip-flop*.

El simulador captura las transiciones de estados y secuencias de salida y las despliega en forma tabular, como formas de onda en el diagrama de tiempos, o como ambas cosas. Por lo general, todos los valores de salida del *flip-flop* se pueden capturar y examinar y, por tanto, se pueden verificar todas las transiciones de estados. En un circuito real, lo común es que sólo sean visibles las salidas externas, y que las variables de estado sean inaccesibles desde los *pines* en el módulo del circuito. En estos casos, hay que deducir el comportamiento del circuito a partir de las secuencias de salida producidas por cada secuencia de prueba de entrada.

Al analizar los resultados de la simulación, se verifica que el estado del circuito sea correcto después de cada transición de reloj. En un circuito de modelo de Mealy, los cambios de salida se pueden iniciar mediante cambios de estado o de entrada y, por tanto, se pueden presentar en cualquier momento dentro del periodo de reloj. En los circuitos de modelo de Moore, las salidas sólo deben cambiar cuando cambien las variables de estado y, por tanto, no deben verse afectadas por transiciones en las señales de entrada.

Análisis de tiempos

El análisis de tiempos implica la estimación de los retardos por propagación, los que determinan el desempeño del circuito, y la verificación de violación de las restricciones de tiempos durante el funcionamiento de un circuito.

El desempeño de un circuito secuencial síncrono casi siempre se especifica en términos de la frecuencia de reloj máxima con que puede operar el circuito, dada por el inverso del periodo mínimo de reloj, $T_{mín}$. $T_{mín}$ está limitado por los retardos por propagación a través de los *flip-flops* y las compuertas de lógica combinatoria del circuito, así como por los requisitos de tiempo para la configuración del *flip-flop*.

Con referencia al modelo de circuito secuencial síncrono general de la figura 8.56, podemos agrupar los diversos retardos por propagación del circuito en dos parámetros: los retardos por propagación por los *flip-flops*, t_{FF}, y los

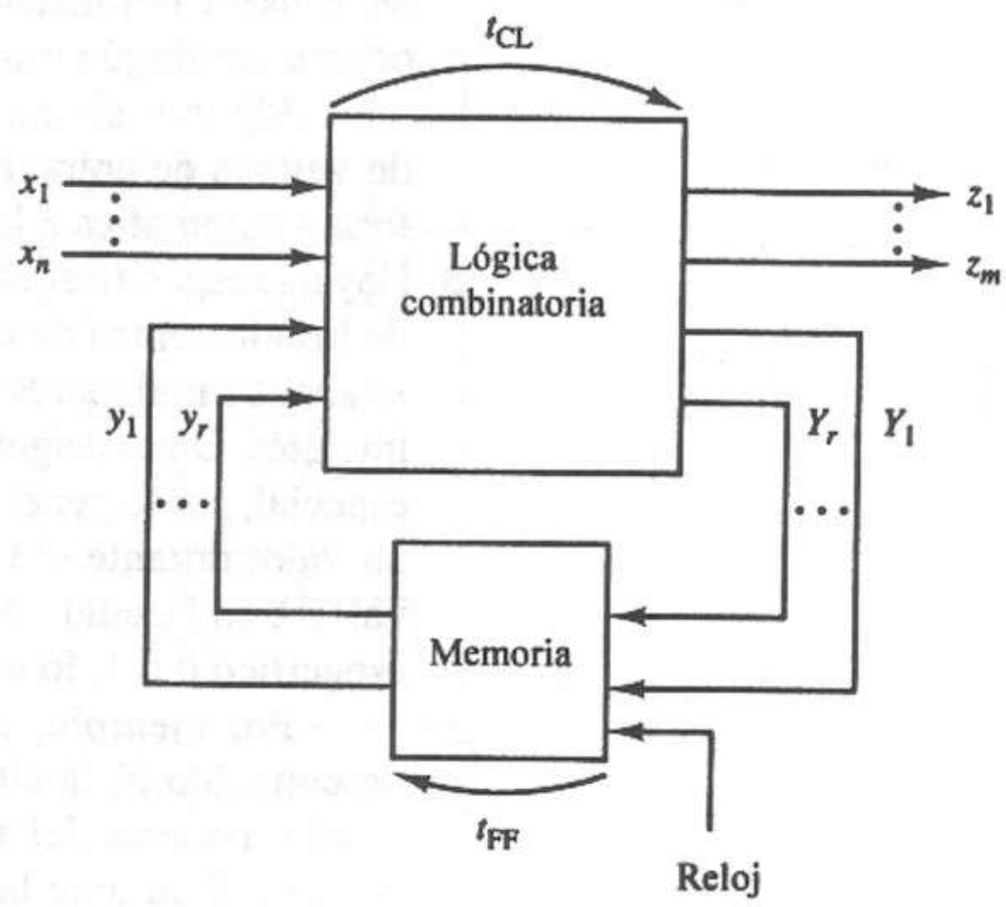


Figura 8.56 Retardos por propagación en un circuito secuencial.

retardos debidos a la lógica combinatoria, t_{CL}. Después de una transición activa del reloj, se propagan nuevos valores a través de los *flip-flop* a las variables secundarias, $y_1 \ldots y_r$, y después estos valores se propagan a través de la lógica combinatoria a las variables de estado siguiente, $Y_1 \ldots Y_r$. Debido a las restricciones en el tiempo de configuración, todas las entradas de excitación de *flip-flop* deben ser estables durante un periodo de tiempo mayor o igual que t_{su} antes de la siguiente transición de reloj. Por tanto, el tiempo mínimo entre transiciones de reloj sucesivas está dado por

$$T_{\text{mín}} \geq t_{FF} + t_{CL} + t_{su}$$

de modo que $T_{\text{mín}}$ representa el periodo de reloj mínimo.

El retardo por propagación t_{FF} se mide desde la ocurrencia de una transición de reloj hasta el tiempo en que todas las salidas del *flip-flop* son estables; por tanto, es función de los parámetros t_{PHL} y t_{PLH} del *flip-flop* descritos en el capítulo 6. Como todos los *flip-flop* están sincronizados con el mismo reloj, t_{FF} se elige como el máximo de los parámetros t_{PHL} y t_{PLH} del *flip-flop* individual a fin de reflejar el retardo en el peor de los casos.

El tiempo de retardo por propagación t_{CL} es función de los parámetros t_{PHL} y t_{PLH} de las compuertas individuales que conforman el bloque lógico combinatorio. El número de compuertas entre cada entrada y salida del bloque lógico combinatorio puede variar. Por ello, calculamos el tiempo de retardo por propagación t_{CL} como el retardo de ruta en el peor de los casos entre las entradas y las salidas de estado siguiente.

Por lo general, las salidas externas de un circuito secuencial no están sujetas a restricciones, como lo están las entradas de excitación de los *flip-flops*. En consecuencia, es común que el tiempo de salida no sea tan crítico. En una máquina Mealy, las salidas son funciones de las entradas externas y de las variables de estado. Por tanto, un cambio en la salida puede ser resultado de un cambio en la entrada, que se propagaría a la salida en el tiempo t_{CL}, o de un cambio en la variable de estado, en cuyo caso el cambio se propagaría hasta la salida después de un retardo de $t_{FF} + t_{CL}$.

En una máquina Moore, las salidas sólo dependen de las variables de estado. Por tanto, todos los cambios de salida se presentarían después de un retardo $t_{FF} + t_{CL}$ medido desde el instante de la transición de reloj.

Con frecuencia se realiza una verificación de restricciones durante la simulación lógica para garantizar que no se violan las restricciones de tiempos para el reloj y las secuencias de entrada dadas. Como definimos en el capítulo 6, cada *flip-flop* tiene un tiempo mínimo de configuración, t_{su}, y un tiempo de retención, t_h, como vemos en la figura 8.57. Para una transición de reloj en el

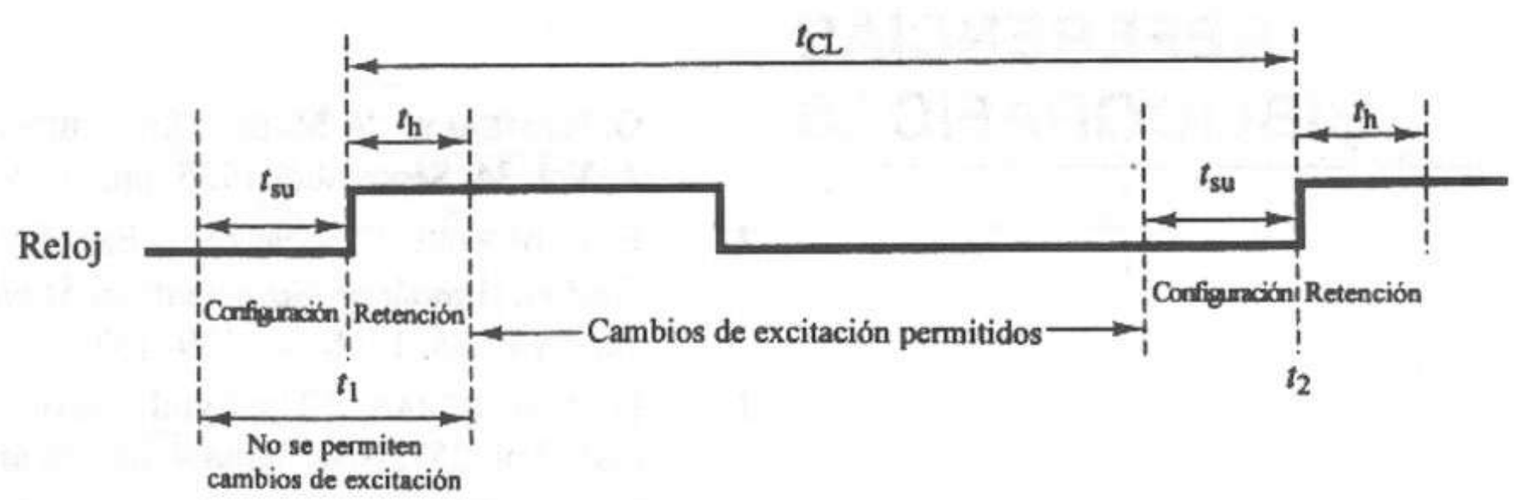


Figura 8.57 Restricciones de tiempos en un circuito secuencial.

instante t_k, no deben presentarse cambios en los valores de entrada de excitación del *flip-flop* dentro del periodo de tiempo $[t_k - t_{su}, t_k + t_h]$, como indicamos en la figura 8.57.

Debemos sincronizar los cambios en las entradas para no violar las restricciones de los tiempos de configuración y retención. En relación con la figura 8.57, para las transiciones de reloj en los instantes t_1 y t_2, los cambios en las entradas de excitación del *flip-flop* sólo están permitidos en el intervalo de tiempo $[t_1 + t_h, t_2 - t_{su}]$. Como existe un retardo de tiempo t_{CL} entre un cambio de entrada y el cambio correspondiente en las variables de estado siguiente $Y_1 \ldots Y_r$, el momento t_x de cualquier cambio de entrada debe satisfacer las restricciones:

$$t_1 + t_h < t_x + t_{CL} < t_2 - t_{su}$$

o

$$t_1 + t_h - t_{CL} < t_x < t_2 - t_{su} - t_{CL}$$

Para verificar las restricciones durante la simulación lógica, hay que especificar los parámetros t_{su} y t_h para cada *flip-flop*. Durante la simulación, cualquier entrada de excitación del *flip-flop* que cambie dentro del periodo de tiempo de t_{su}, antes de la transición de reloj, hasta *t*h, después de la transición de reloj, es señalada por el simulador como un error en la salida de simulación. En algunos casos, el simulador puede dar un valor indeterminado *X* al estado de un *flip-flop* para prevenir al diseñador de un problema potencial. Este uso del valor indeterminado indica la existencia de un posible estado desconocido en el *flip-flop* debido a la violación de las restricciones de los tiempos de configuración y de retención.

8.6 Resumen

En este capítulo presentamos procedimientos para el análisis de un circuito secuencial síncrono, dado su diagrama lógico, tabla de estados o diagrama de estados, así como los procedimientos para realizar circuitos lógicos secuenciales síncronos con especificación completa e incompleta. Prestamos atención a las tablas de entrada de los *flip-flops* y a la generación de mapas de excitación, así como al desarrollo de diagramas de estados y tablas de estados a partir de la descripción verbal del problema. Completamos muchos ejemplos de diseño para ilustrar las técnicas de síntesis. El lector tendrá ahora una idea del problema de la síntesis de circuitos secuenciales síncronos. En el capítulo 9 examinaremos los métodos para optimizar circuitos secuenciales síncronos. Si el lector desea obtener información más detallada acerca de problemas de síntesis, lo invitamos a que consulte las referencias que presentamos a continuación.

REFERENCIAS BIBLIOGRÁFICAS

1. G. H. MEALY, "A Method for Synthesizing Sequential Circuits," *Bell Sys. Tech. J.*, Vol. 34, September 1955, pp. 1045–1079.
2. E. F. MOORE, "Gedanken—Experiments on Sequential Machines," *Automata Studies, Annals of Mathematical Studies*, No. 34. Princeton, NJ: Princeton University Press, 1956, pp. 129–153.
3. D. A. HUFFMAN, "The Synthesis of Sequential Switching Circuits," *J. Franklin Inst.*, Vol. 257, Nos. 3 and 4, March and April 1954, pp. 161–190, 275–303.
4. E. J. MCCLUSKEY, *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965.

5. ZVI KOHAVI, *Switching and Finite Automata Theory.* New York: McGraw-Hill Book Co., 1970.
6. TAYLOR L. BOOTH, *Digital Networks and Computer Systems.* New York: Wiley, 1971.
7. J. F. WAKERLY, *Digital Design Principles and Practices*, 2nd Ed., Chapter 2. Englewood Cliffs, NJ: Prentice Hall, 1994.
8. J. P. HAYES, *Introduction to Digital Logic Design*, Reading, MA: Addison-Wesley, 1993.
9. R. H. KATZ, *Contemporary Logic Design*, Menlo Park, CA: Benjamin/Cummings Publishing Co., 1994.
10. J. R. ARMSTRONG, *Chip-level Modeling with VHDL.* Englewood Cliffs, NJ: Prentice Hall, 1989.
11. *AUTOLOGIC VHDL REFERENCE MANUAL*, Mentor Graphics Corp., 8005 S.W. Boeckman Rd., Wilsonville, OR 97070, April 1992.
12. T. K. EDWARDS, *Automatic Logic Synthesis for Digital Systems*, New York: McGraw-Hill Book Co., 1992.

PROBLEMAS

8.1 Para el circuito secuencial síncrono de la figura P8.1, determine:

(a) La tabla de estados mediante mapas K y $A \equiv 0$, $B \equiv 1$.

(b) El diagrama de estados, si la entrada del circuito está en forma de pulso.

(c) El diagrama de tiempos para una secuencia de entrada $x = 00100110$ y el estado inicial $y^0 = 1$.

8.2 Dado el circuito secuencial síncrono de la figura P8.2 con entradas de nivel:

(a) Trace un diagrama de tiempos para $x = 000101011$ y $y^0 = 0$.

(b) Determine el diagrama de estados.

(c) Determine la tabla de estados.

8.3 Para el circuito secuencial de la figura P8.3, determinar:

(a) La tabla de estados ($A \equiv 0$, $B \equiv 1$).

(b) El diagrama de estados.

(c) Un diagrama de tiempos, si el estado inicial es $y^0 = 0$ y $x = 001011000$. Este circuito es síncrono de nivel.

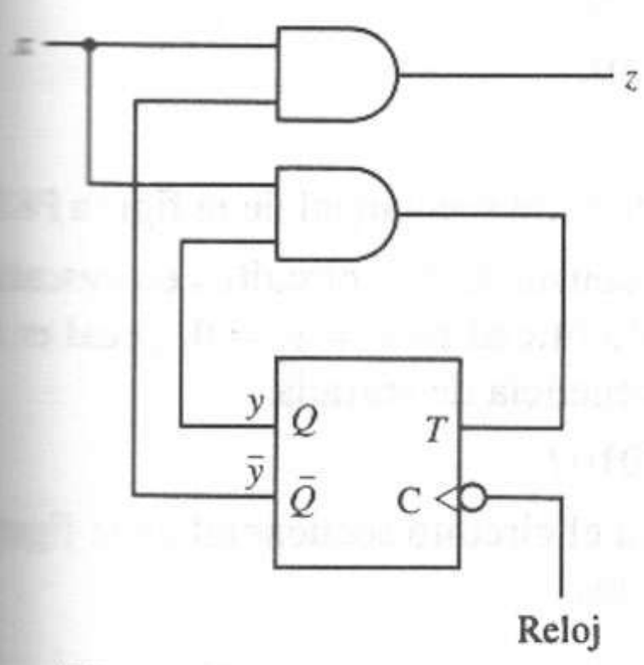


Figura P8.1

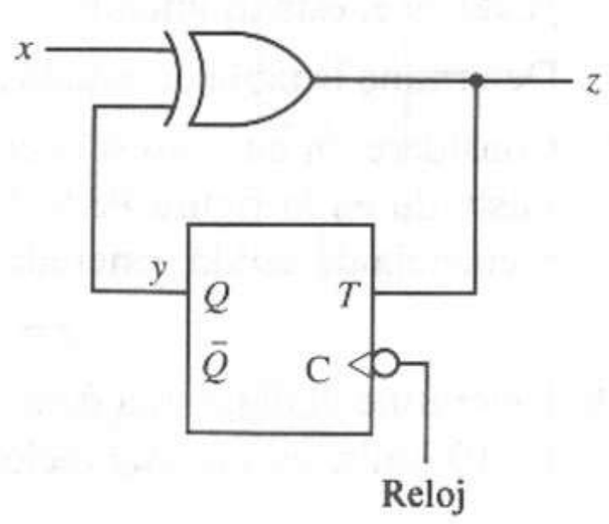


Figura P8.2

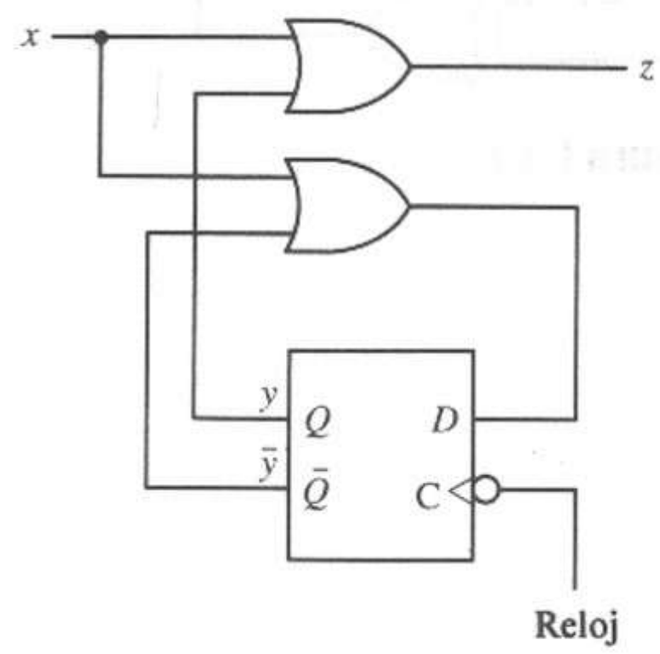


Figura P8.3

8.4 Trace el diagrama lógico para un circuito secuencial síncrono que utilice *flip-flops* T con reloj y las funciones de conmutación

$$z = T_1 = x\bar{y}_2$$
$$T_2 = x \oplus y_1$$

Determine un diagrama de estados del circuito con la asignación

	y_1	y_2
A:	0	0
B:	0	1
C:	1	1
D:	1	0

8.5 Trace el diagrama lógico para la implantación de un circuito secuencial mediante un *flip-flop* D con reloj, utilizando las ecuaciones lógicas

$$Y_1 = \bar{x} \oplus y_1 = x \odot y_1$$
$$Y_2 = x + y_1 + y_2$$
$$z = xy_1\bar{y}_2$$

Determine una tabla de estados binarios para este circuito.

8.6 Analice el circuito secuencial síncrono de la figura P8.6. Suponga que las entradas son niveles binarios y que se utiliza la siguiente asignación de estados:

	y_1	y_2
A:	0	0
B:	0	1
C:	1	1
D:	1	0

Utilice mapas K para determinar:

(a) La tabla de estados. **(b)** El diagrama de estados.

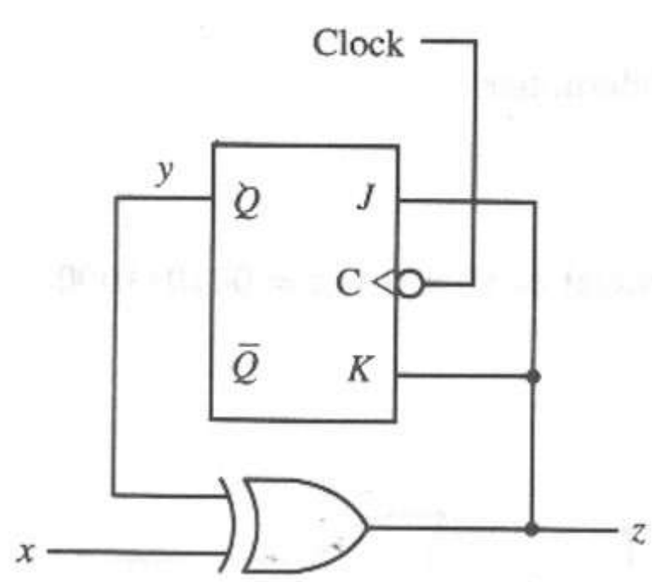


Figura P8.7

8.7 Si el circuito secuencial de la figura P8.7 produce una secuencia de salida

$$z = 11011111$$

cuando aplicamos la secuencia de entrada

$$x = 01101010,$$

¿cuál es el estado inicial?

8.8 Determine la tabla de estados para el circuito secuencial de la figura P8.8.

8.9 Considere un circuito secuencial que consta de dos circuitos en cascada ilustrado en la figura P8.9. Si el estado inicial es $y_1 = y_2 = 0$, ¿cuál es la secuencia de salida generada por la secuencia de entrada

$$x = 0110111010?$$

8.10 Determine el diagrama de estados para el circuito secuencial de la figura P8.10 utilizando la asignación de estados

	y_1	y_2
A:	0	0
B:	0	1
C:	1	1
D:	1	0

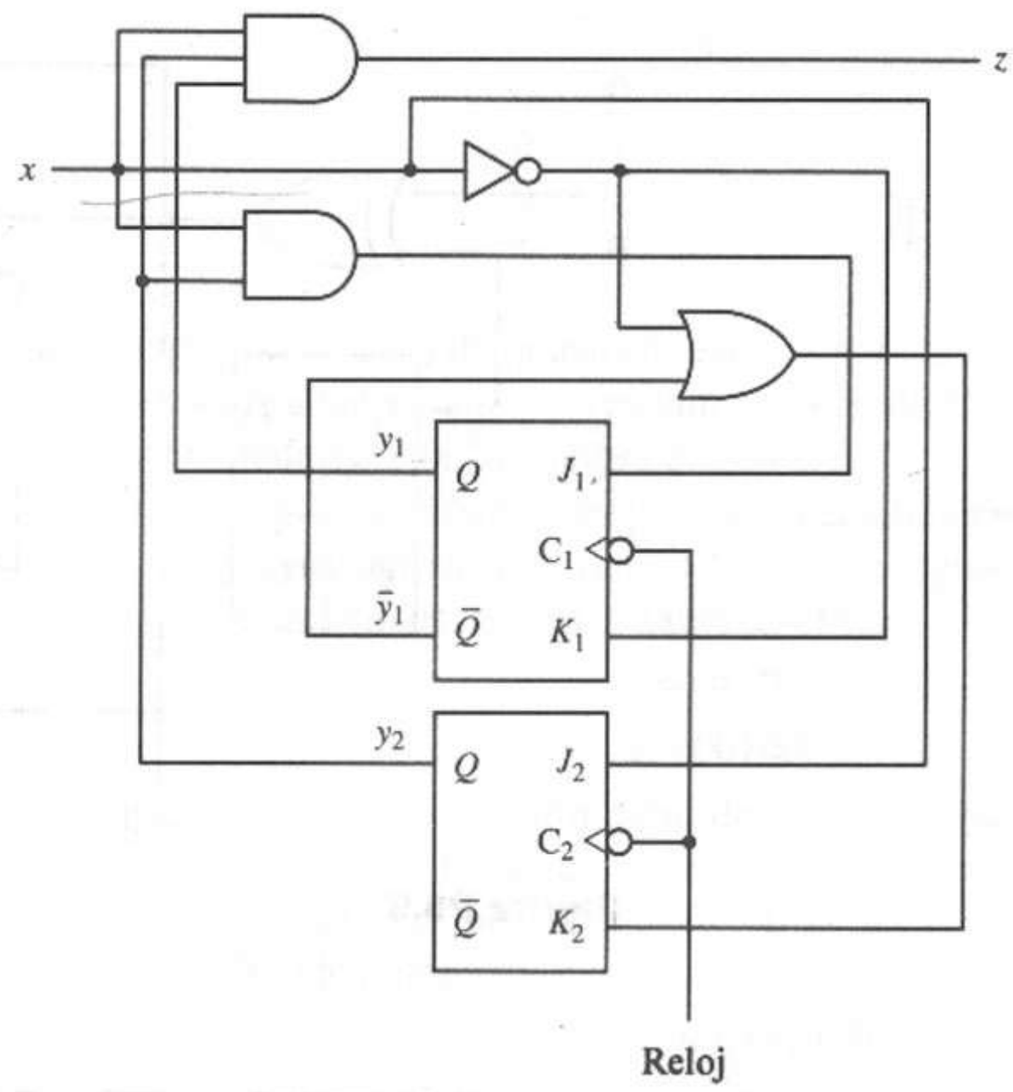


Figura P8.6

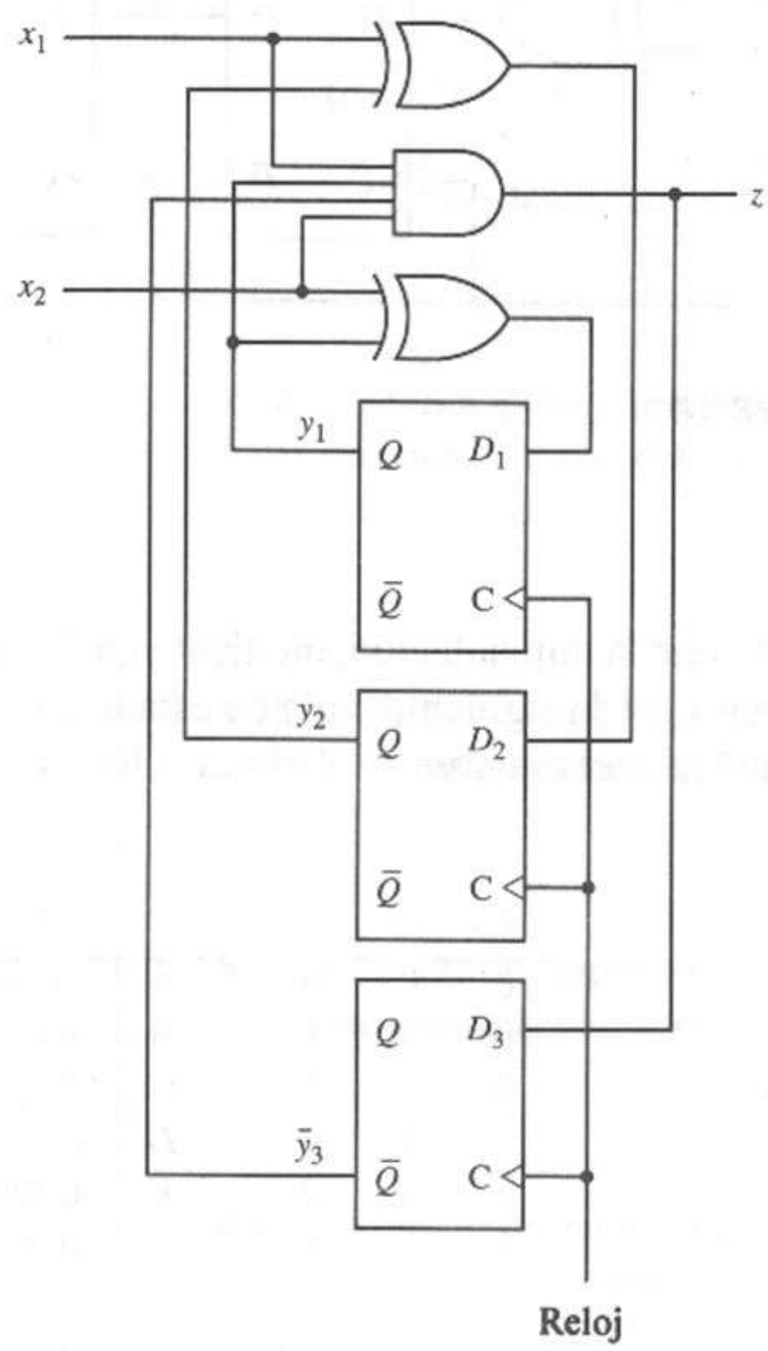


Figura P8.8

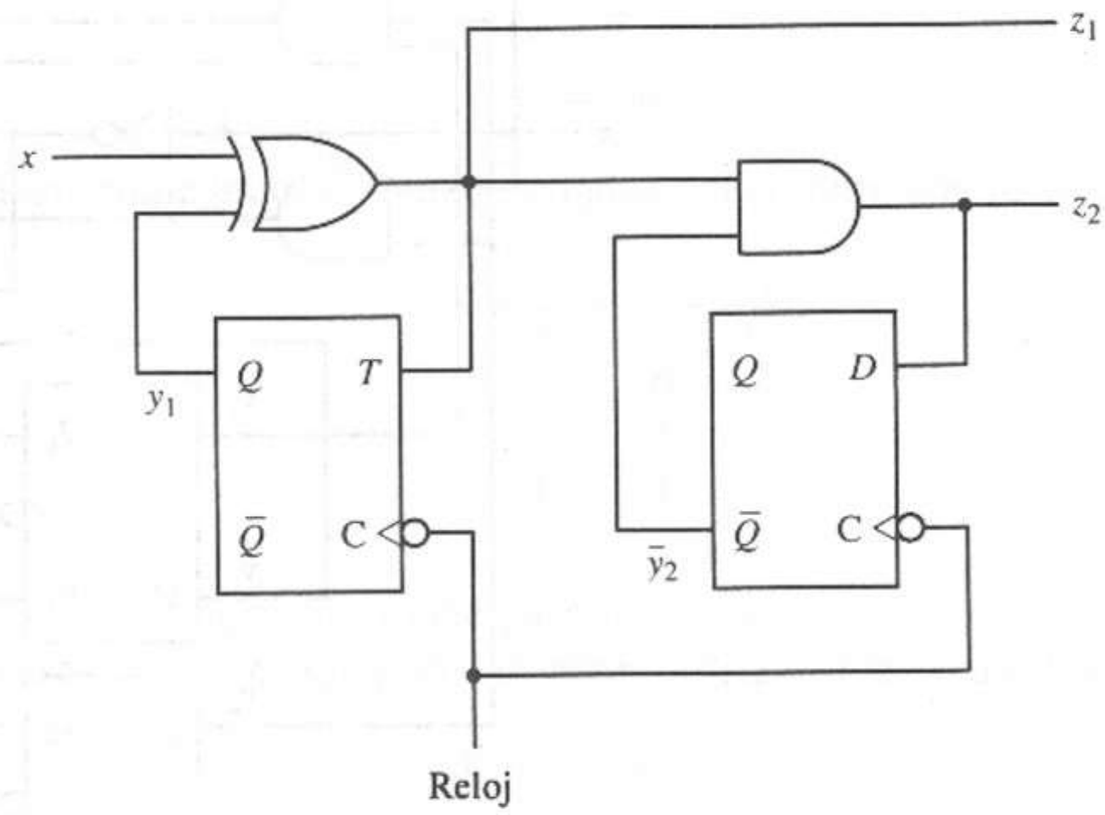


Figura P8.9

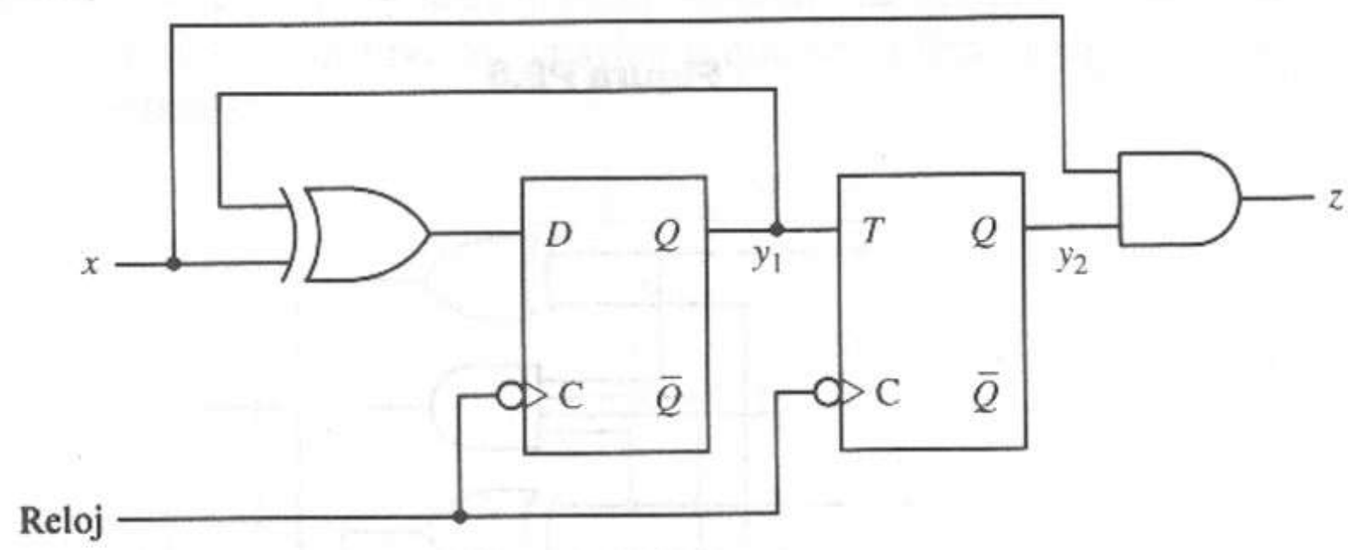


Figura P8.10

8.11 Determine la implantación, mediante un *flip-flop* D, del circuito secuencial definido por la siguiente tabla de estados. Utilice la asignación de estados dada. Trace el diagrama del circuito lógico.

y_3	y_2	y_1
0	0	0
0	0	1
0	1	1
0	1	0
1	0	0
1	0	1

	x = 0	x = 1
A	$D/0$	$C/0$
B	$E/0$	$A/1$
C	$F/1$	$B/0$
D	$A/1$	$F/1$
E	$C/0$	$E/0$
F	$B/0$	$D/1$

8.12 Obtenga una realización mediante un *flip-flop* D del circuito secuencial síncrono especificado por la siguiente tabla de estados. Utilice la asignación de estados dada. Escriba las ecuaciones lógicas combinatorias.

y_3	y_2	y_1		x = 0	x = 1
0	0	0	A	$B/0$	$E/0$
0	0	1	B	$A/1$	$C/1$
0	1	0	C	$B/0$	$C/1$
0	1	1	D	$C/0$	$E/0$
1	0	0	E	$D/1$	$A/0$

8.13 Determine cuatro diagramas de estados para los circuitos secuenciales síncronos especificados por por los siguientes requisitos. Cada circuito tiene una única línea de entrada x y una única línea de salida z.

(a) El primer circuito debe producir una salida $z = 1$ cuando ocurran dos entradas x consecutivas con valor 1 lógico. La siguiente entrada posterior a los dos unos lógicos regresa la salida a 0 lógico. Por ejemplo,

$$x = 01100111110$$
$$z = 00100010100$$

(b) El segundo circuito debe detectar la secuencia de entrada 101, produciendo $z = 1$ cuando se presente el último 1. La salida z regresa a 0 en el siguiente pulso de reloj. Dos secuencias 101 se pueden traslapar. Por ejemplo,

$$x = 010101101$$
$$z = 000101001$$

(c) Repita el problema 8.13b, pero sin permitir traslapes. Por ejemplo,

$$x = 010101101$$
$$z = 000100001$$

(d) El cuarto circuito detecta una secuencia 01. La secuencia hace que $z = 1$, y z sólo regresa a 0 cuando ocurre una secuencia de entrada 00. Para los demás casos, $z = 0$. Por ejemplo,

$$x = 010100100$$
$$z = 011110110$$

8.14 Deduzca el diagrama de estados mínimo de un circuito secuencial con reloj que reconozca la secuencia de entrada 1010. Las secuencias se pueden traslapar. Por ejemplo,

$$x = 001010010101011 10$$
$$z = 000001000010100 00$$

8.15 Determine la tabla de estados de un circuito secuencial síncrono que detecte la secuencia de entrada 0101. Las secuencias se pueden traslapar, como sigue:

$$x = 010101001101011$$
$$z = 000101000000010$$

8.16 Obtenga un diagrama de estados mínimo para un circuito secuencial con reloj que reconozca la secuencia de entrada 1001, permitiendo los traslapes. Por ejemplo:

$$x = 01010010001100 10010$$
$$z = 00000010000000 10010$$

8.17 Deduzca las ecuaciones lógicas que implanten el circuito secuencial de cuatro estados definido por la siguiente tabla de estados, utilizando la asignación de estados dada y:

(a) *Flip-flops* D.

(b) *Flip-flops* JK con reloj.

(c) *Flip-flops* SR con reloj.

y_1	y_2		x = 0	x = 1
0	0	A	$B/0$	$C/0$
0	1	B	$D/0$	$A/1$
1	1	C	$A/1$	$D/0$
1	0	D	$D/1$	$B/1$

8.18 Para el siguiente circuito con la asignación de estados dada, determine una implantación mediante *flip-flops* JK con reloj. Escriba las ecuaciones lógicas y trace el diagrama lógico.

y_1	y_2		x = 0	x = 1
0	0	A	$B/0$	$D/0$
0	1	B	$C/0$	$A/0$
1	0	C	$D/0$	$B/0$
1	1	D	$A/1$	$C/1$

8.19 Implante el circuito del problema 8.18 mediante *flip-flops* T con reloj.

8.20 Implante el circuito del problema 8.18 mediante *flip-flops* D.

8.21 Dadas las siguientes tabla de estados reducida y asignación, determine las ecuaciones lógicas y el diagrama lógico.

(a) Usando *flip-flops* D. **(b)** Usando *flip-flops* JK con reloj.

y_1	y_2		x = 0	x = 1
0	0	A	$A/0$	$B/0$
0	1	B	$C/0$	$B/0$
1	1	C	$D/0$	$B/0$
1	0	D	$A/1$	$B/0$

8.22 Determine el diagrama lógico para la implantación del siguiente circuito secuencial, dada la asignación de estados y:

(a) *Flip-flops* D. **(c)** *Flip-flops* T con reloj.

(b) *Flip-flops* JK con reloj.

y_1	y_2		x = 0	x = 1
0	0	A	$A/0$	$B/0$
0	1	B	$C/0$	$B/0$
1	1	C	$D/0$	$B/0$
1	0	D	$B/1$	$A/0$

8.23 Determine una realización mediante *flip-flops* JK con reloj para las siguientes tabla de estados reducida y asignación:

y_1	y_2	y_3		x = 0	x = 1
0	0	0	A	$B/0$	$D/0$
1	0	1	B	$A/0$	$C/1$
1	0	0	C	$D/1$	$C/0$
0	0	1	D	$B/1$	$E/1$
0	1	0	E	$C/0$	$A/0$
1	1	0	F	$E/0$	$F/1$

8.24 Diseñe un contador arriba/abajo de 2 bits, módulo 3, con la siguiente tabla funcional, usando *flip-flops* JK.

s_1	s_0	Modo
0	0	Arriba
0	1	Abajo
1	0	Módulo 3
1	1	Módulo 3

8.25 Utilice *flip-flops* D para diseñar un contador generador de números seudoaleatorios de 3 bits. El circuito tiene una entrada de control x. Cuando $x = 0$, el circuito debe operar como un contador binario hacia arriba. En caso contrario, debe operar como un generador de números seudoaleatorios, según la siguiente tabla funcional.

Estado actual	Contador binario hacia arriba $x = 0$	Generador de números seudoaleatorios $x = 1$
0	1	0
1	2	4
2	3	5
3	4	1
4	5	2
5	6	6
6	7	7
7	0	3

Estado siguiente

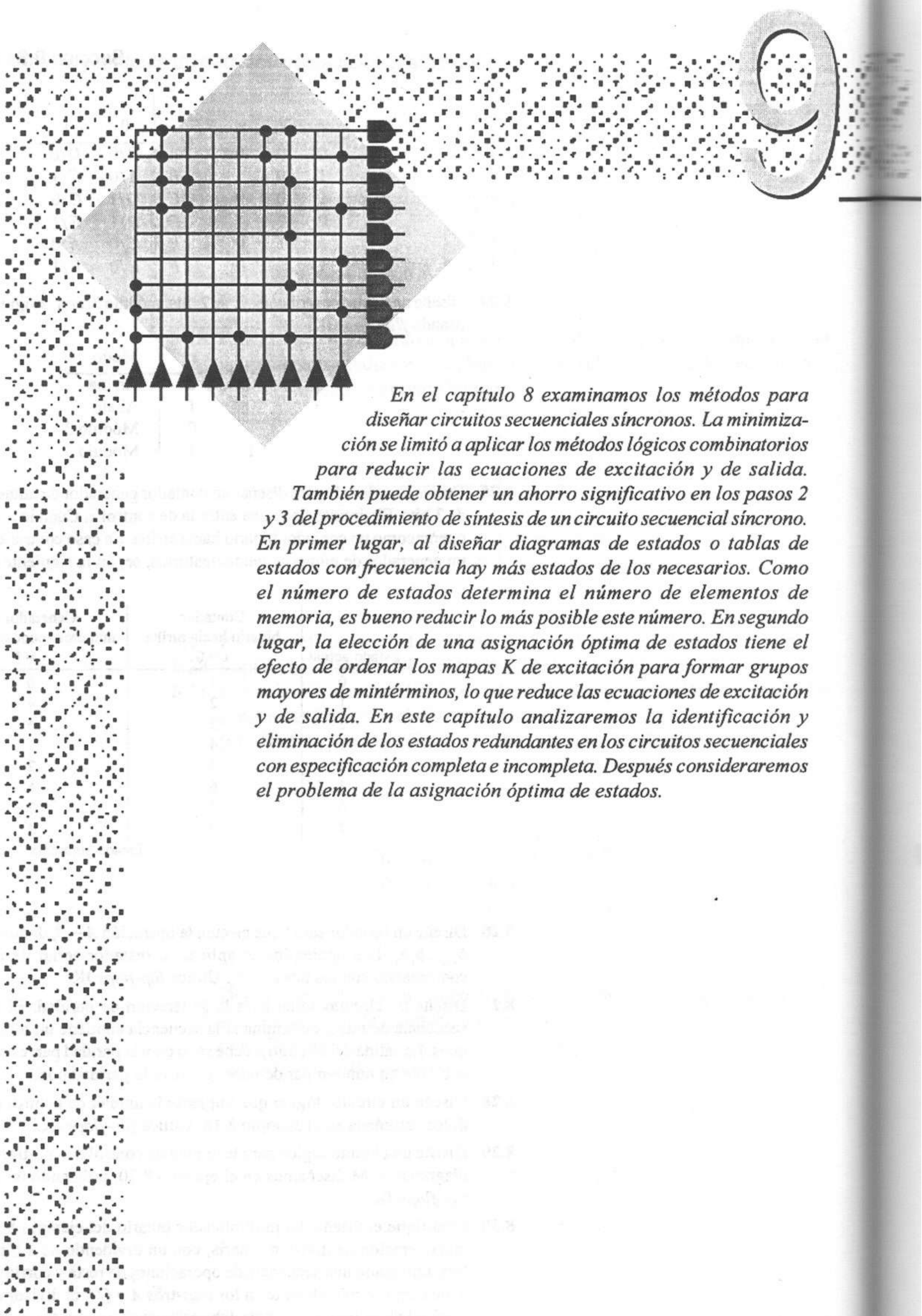
8.26 Diseñe un restador serial que ejecute la operación $A - B$, donde $A = {}_{an-1}...a_1a_0$ y $B = b_{n-1}...b_1b_0$. Los operandos se aplican al restador serial de manera secuencial, comenzando con los bits a_0 y b_0. Utilice *flip-flops* JK.

8.27 Diseñe un circuito serial para la generación de paridad. El circuito recibe una secuencia de bits y determina si la secuencia contiene un número par o impar de unos. La salida del circuito p debe ser 0 para la paridad par; es decir, si la secuencia contiene un número par de unos, y 1 para la paridad non.

8.28 Diseñe un circuito lógico que implante la unidad de control para la máquina de dulces diseñada en el ejemplo 8.18. Utilice *flip-flops* JK.

8.29 Diseñe un circuito lógico para la unidad de control del multiplicador binario cuyo diagrama ASM diseñamos en el ejemplo 8.20, utilizando un número mínimo de *flip-flops* JK.

8.30 Modifique el diseño del multiplicador binario del ejemplo 8.20 para que realice una operación de división binaria, con un dividendo de 8 bits y un divisor de 4 bits, utilizando una secuencia de operaciones de resta y corrimiento. El dividendo debe cargarse inicialmente en los registros A y Q y el divisor en el registro M. Al final del algoritmo, el cociente debe estar en el registro Q y el residuo en el registro A.



En el capítulo 8 examinamos los métodos para diseñar circuitos secuenciales síncronos. La minimización se limitó a aplicar los métodos lógicos combinatorios para reducir las ecuaciones de excitación y de salida. También puede obtener un ahorro significativo en los pasos 2 y 3 del procedimiento de síntesis de un circuito secuencial síncrono. En primer lugar, al diseñar diagramas de estados o tablas de estados con frecuencia hay más estados de los necesarios. Como el número de estados determina el número de elementos de memoria, es bueno reducir lo más posible este número. En segundo lugar, la elección de una asignación óptima de estados tiene el efecto de ordenar los mapas K de excitación para formar grupos mayores de mintérminos, lo que reduce las ecuaciones de excitación y de salida. En este capítulo analizaremos la identificación y eliminación de los estados redundantes en los circuitos secuenciales con especificación completa e incompleta. Después consideraremos el problema de la asignación óptima de estados.

Simplificación de circuitos secuenciales

9.1 Estados redundantes

En un sentido general, decimos que dos estados son *equivalentes* si no podemos distinguir entre ellos. En otras palabras, no podemos determinar en cuál de dos estados equivalentes inicia un circuito secuencial aplicando entradas y observando las salidas. Si esta condición ocurre para toda secuencia de entrada, uno de estos estados es redundante y se puede eliminar sin alterar el comportamiento del circuito.

Los estados redundantes surgen por lo general en las primeras fases del diseño, cuando una descripción verbal de la función del circuito secuencial se transforma en un diagrama o tabla de estados. La eliminación de los estados redundantes es importante por varias razones:

1. *Costo*: La cantidad de elementos de memoria está relacionada directamente con el número de estados.

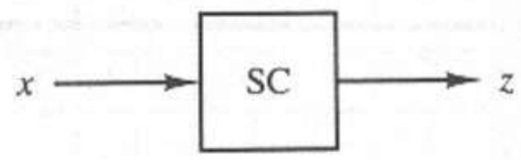
2. *Complejidad*: Cuantos más estados tenga el circuito, más complejos serán el diseño y su implantación asociada.

3. *Ayuda para el análisis de fallas*: Con frecuencia, las rutinas de diagnóstico suponen que no existen estados redundantes.

9.1.1 Equivalencia de estados

Introduciremos la idea de equivalencia de estados por medio de un sencillo ejemplo. Consideremos el circuito de las figuras 9.1a y b. Supongamos que se desconoce el estado inicial. Si aplicamos una entrada $x = 0$ y la salida es $z = 1$, lo único que sabemos del estado inicial es que es A o B o C. De manera análoga, si la salida es $z = 0$ cuando se aplica la entrada $x = 0$, el estado inicial es D o E. Observe que obtenemos una conclusión similar para la entrada $x = 1$. Por tanto, concluimos que los estados A, B y C son equivalentes y que los estados D y E son equivalentes para una secuencia de entrada de longitud 1, es decir, *1-equivalentes*. Las figuras 9.1c y d muestran el comportamiento para las secuencias de entrada de longitudes 2 y 3. Observe que los estados B y C y los estados D y E son 2-equivalentes. Los estados B y C también son 3-equivalentes, y de hecho se puede mostrar que estos dos estados son K-equivalentes para toda K.

Con estos hechos como base, definiremos en forma precisa lo que se entiende por estados equivalentes.



(a)

	x = 0	x = 1
A	C/1	B/0
B	C/1	E/0
C	B/1	E/0
D	D/0	B/1
E	E/0	A/1

(b)

Estado inicial	Secuencias de entrada 00	01	11	10
A	11	10	01	00
B	11	10	00	01
C	11	10	00	01
D	00	01	11	10
E	00	01	11	10

(c)

Estado inicial	Secuencias de entrada 000	001	010	011	100	101	110	111
A	111	110	100	101	011	010	000	001
B	111	110	100	101	000	001	011	010
C	111	110	100	101	000	001	011	010
D	000	001	011	010	111	110	100	101
E	000	001	011	010	111	110	101	100

(d)

Figura 9.1 Estados redundantes. **(a)** Circuito secuencial *SC*. **(b)** Tabla de estados. **(c)** Secuencias de salida de longitud 2. **(d)** Secuencias de salida de longitud 3.

Definición

Los estados $S_1, S_2, \ldots, S_j$ de un circuito secuencial son *equivalentes* si y sólo si, para cada secuencia de entrada posible, se obtiene la misma secuencia de salida, sin importar cuál de los estados $S_1, S_2, \ldots, S_j$ sea el estado inicial.

Podemos enunciar esta definición de otra manera para pares de estados. Sean S_k y S_l los estados siguientes del circuito secuencial *SC* cuando se aplica la entrada I_p mientras el circuito se encuentra en los estados S_i y S_j, respectivamente. Entonces S_i y S_j son *equivalentes* si y sólo si para cada entrada posible I_p,

1. La salida producida por el estado S_i es igual a la salida producida por el estado S_j.
2. Los estados siguientes S_k y S_l son equivalentes.

Podemos deducir la segunda definición de la primera, de la manera siguiente. Si S_i produce una salida diferente a la salida de S_j para una entrada I_p, entonces S_i y S_j no son equivalentes. Por tanto, la primera condición es necesaria. Si S_k y S_l no son equivalentes, existe una secuencia de entrada $I_1 I_2 \ldots I_k$ que produce una secuencia de salida diferente con S_i como estado inicial y con S_j como estado inicial. Por tanto, $I_p I_1 I_2 \ldots I_k$ producirá una secuencia de salida diferente con S_i como estado inicial que con S_j como estado inicial. Así, S_i y S_j no pueden ser equivalentes, a menos que se satisfaga la segunda condición. Por último, es

claro que las condiciones son suficientes para que S_i y S_j sean equivalentes, de modo que las dos definiciones son sinónimas. Estas dos condiciones forman la base para todas las técnicas de reducción de estados.

9.1.2 Equivalencia y relaciones de compatibilidad

Sean x y y elementos de un conjunto S. Supongamos que x y y están relacionados mediante una propiedad r, lo que denotamos con $x\ r\ y$. Una *relación* R en el conjunto S es el conjunto de todos los pares ordenados (s_i,s_j) tales que s_i y s_j son elementos de S y tales que $s_i\ r\ s_j$. R es *reflexiva* si y sólo si $s_i\ r\ s_j$ para toda s_i en S. R es *simétrica* si y sólo si $s_i\ r\ s_j$ implica que $s_j\ r\ s_i$. R es *transitiva* si y sólo si $s_i\ r\ s_j$ y $s_j\ r\ s_k$ implican que $s_i\ r\ s_k$. Una *relación de equivalencia* en S es una relación en S que es simétrica, reflexiva y transitiva. Podemos utilizar una relación de equivalencia para dividir los elementos de S en subconjuntos ajenos llamados *clases de equivalencia.*

Se puede demostrar que la equivalencia de estados define una relación de equivalencia en el conjunto de estados de un circuito secuencial con especificación completa. Por tanto, utilizamos las clases de equivalencia para definir los estados de la tabla de estados reducida.

Una relación es S es una *relación de compatibilidad* si y sólo si la relación es reflexiva y simétrica. Las relaciones de compatibilidad definen subconjuntos de S conocidos como *clases de compatibilidad.* Por lo general, estos subconjuntos no son ajenos. El tema de las clases de compatibilidad será importante cuando analicemos en una sección posterior de este capítulo la reducción de las tablas de estado con especificación incompleta. El lector puede consultar las referencias [1–3] si desea un análisis más detallado de las relaciones de equivalencia y compatibilidad.

9.2 Reducción de estados en circuitos con especificación completa

Ahora presentaremos tres técnicas para determinar estados equivalentes en los circuitos secuenciales con especificación completa:

1. Inspección
2. Partición
3. La tabla de implicación

En cada caso, utilizaremos la técnica para determinar las clases de equivalencia del circuito y después eliminaremos todos los estados de cada clase de equivalencia, menos uno, a fin de reducir la tabla de estados.

9.2.1 Inspección

La técnica más sencilla y más obvia es la de reconocer los estados equivalentes por inspección. En este método sólo debemos reconocer varias filas en la tabla de estados que realicen la misma función, para después eliminar los estados redundantes.

EJEMPLO 9.1

En este ejemplo hay tres ejemplos de filas equivalentes. En primer lugar, examine el circuito secuencial definido mediante la tabla de estados de la figura 9.2a. Ésta se puede reducir por inspección, observando que los estados *B* y *D* realizan exactamente la misma función. Por tanto, se puede eliminar el estado *D* de la tabla, para lo cual basta con eliminar la fila *D* de la tabla y remplazar el estado D por su estado equivalente *B*. Este procedimiento produce la tabla de estados de la figura 9.2b.

	x = 0	x = 1
A	B/0	C/1
B	C/0	A/1
C	D/1	B/0
D	C/0	A/1

(a)

	x = 0	x = 1
A	B/0	C/1
B	C/0	A/1
C	B/1	B/0

(b)

	x = 0	x = 1
A	B/0	C/1
B	B/0	A/1
C	D/1	B/0
D	D/0	A/1

(c)

	x = 0	x = 1
A	B/0	C/1
B	B/0	A/1
C	B/1	B/0

(d)

	x = 0	x = 1
A	B/0	C/1
B	D/0	A/1
C	D/1	B/0
D	B/0	A/1

(e)

Figura 9.2 Equivalencia de estados por inspección. **(a)** Circuito 1. **(b)** Circuito 1 reducido. **(c)** Circuito 2. **(d)** Circuito 2 reducido. **(e)** Circuito 3.

En segundo lugar, analicemos el circuito secuencial de la figura 9.2c. Si el circuito se encuentra en el estado *B* y recibe una entrada 0 lógico, permanecerá en el mismo estado (regresará a sí mismo en un diagrama de estados) con 0 como salida, y si recibe una entrada 1 lógico pasará al estado *A* con una salida 1. Por otro lado, si el circuito se encuentra en el estado *D*, podríamos afirmar lo mismo:

con una entrada 0 lógico permanece en el mismo estado con una salida 0, y con una entrada 1 lógico pasa al estado A con una salida 1. Por tanto, los estados B y D son equivalentes por inspección. La figura 9.2d muestra la tabla de estados reducida.

Nuestro tercer ejemplo aparece en la figura 9.2e. Este ejemplo recuerda al circuito 2; sin embargo, hemos intercambiado los estados siguientes de B y D en la columna $x = 0$. Ahora, al analizar el comportamiento de los estados B y D, observamos que si combinamos el par de estados, el comportamiento con entrada 0 requiere que el circuito secuencial permanezca en el mismo estado (que regrese a sí mismo) con una salida 0 lógico, y con entrada $x = 1$ el circuito secuencial pasará al estado A con una salida 1 lógico. Así, el circuito reducido de la figura 9.2d también es válido para el circuito 3. Éste es otro caso de reducción de estados por inspección.

En resumen, dos estados son equivalentes por inspección cuando las filas de estado siguiente son idénticas, o cuando son idénticas con la excepción de las entradas de "retorno a sí mismo".

9.2.2 Partición

El método de partición implica la determinación sucesiva de particiones P_K, $K = 1, 2, 3,..., l$, donde cada P_K está compuesta por varios *bloques*, cada uno de los cuales consta de un grupo de uno o más estados. Los estados contenidos dentro de un bloque dado de P_K son K-equivalentes. En otras palabras, dado un circuito secuencial con estados S_1, S_2, S_3, S_4, S_5, si $P_K = (S_1S_3)(S_2S_4)(S_5)$, entonces P_K contiene tres bloques y S_1 y S_3 son K-equivalentes, al igual que S_2 y S_4. S_5 no es K-equivalente a algún otro estado del circuito secuencial. Por claridad, utilizaremos el circuito secuencial de la figura 9.1b como ejemplo para describir el procedimiento de partición.

Procedimiento de partición

Paso 1. Formamos la primera partición P_1 colocando dos o más estados en el mismo bloque de P_1 si y sólo si su *salida* es idéntica para cada entrada. Para el ejemplo de la figura 9.1b, $P_1 = (ABC)(DE)$ y, por tanto, los estados dentro de cada bloque son 1-equivalentes. Este paso garantiza que cada bloque de P_1 satisface la condición 1 para los estados equivalentes.

Paso 2. Obtenemos las particiones P_K, $K = 2, 3, 4,..., l$ de manera sucesiva, colocando dos o más estados en el mismo bloque de P_K si y sólo si para cada valor de entrada sus estados siguientes están en un *único* bloque de P_{K-1}. Este proceso iterativo es sugerido por la condición 2 para los estados equivalentes.

Paso 3. Cuando $P_{k+1} = P_K$, es decir, una vez que la partición se repite, los estados de cada bloque de P_K que son K-equivalentes son $(K + 1)$-equivalentes, $(K + 2)$-equivalentes, etcétera, y decimos que P_K es una partición de *equivalencia*. En nuestro ejemplo, una rápida verificación indica que $P_4 = P_3$ y, por tanto, los estados B y C son K-equivalentes para toda K; es decir, son equivalentes. P_K satisface entonces la condición 2 para los estados equivalentes.

Al seguir este procedimiento en nuestro ejemplo, debemos revisar los grupos de estados en cada bloque de P_1. En el primer bloque, los estados siguientes para A, B y C con $x = 0$ están todos en el mismo bloque de P_1. Sin embargo, para $x = 1$, el estado siguiente de A está en un bloque de P_1 distinto del bloque en que se encuentran los estados siguientes de B y C. Por tanto, el bloque (ABC) contenido en P_1 se divide en los bloques $(A)(BC)$ en P_2. Ver la figura 9.3. Los estados siguientes de los estados D y E están en el mismo bloque de P_1 para $x = 0$ y $x = 1$, por lo que D y E permanecerán en el mismo bloque de P_2. Así,

$$P_2 = (A)(BC)(DE)$$

y los estados dentro de cada bloque son 2-equivalentes. Entonces, P_2 debe corresponder exactamente a la figura 9.1c.

Obtenemos la partición P_3 examinando cada bloque de P_2. Los estados siguientes de B y C se encuentran en el mismo bloque de P_2 para cada entrada, por lo que el bloque (BC) permanece intacto en P_3. Sin embargo, los estados siguientes de D y E con $x = 1$ están en bloques diferentes de P_2, por lo que estos dos estados deben aparecer en bloques distintos de P_3. Por tanto, $P_3 = (A)(BC)(D)(E)$. Esto concuerda con la figura 9.1d, así que sólo los estados B y C son 3-equivalentes.

Este procedimiento de obtención de particiones sucesivas se repite hasta lograr la condición establecida en el paso 3.

	Bloques de partición				Acción
Partición P_0	$(ABCDE)$				
Salida para $x = 0$	11100				Separar (ABC) y (DE)
Salida para $x = 1$	00011				Separar (ABC) y (DE)
Partición P_1	(ABC)		(DE)		
Estado siguiente para $x = 0$	CCB		DE		
Estado siguiente para $x = 1$	BEE		BA		Separar (A) y (BC)
Partición P_2	(A)	(BC)	(DE)		
Estado siguiente para $x = 0$	C	CB	DE		
Estado siguiente para $x = 1$	B	EE	BA		Separar (D) y (E)
Partición P_3	(A)	(BC)	(D)	(E)	
Estado siguiente para $x = 0$	C	CB	D	E	
Estado siguiente para $x = 1$	B	EE	B	A	
Partición $P_4 = P_3$	(A)	(BC)	(D)	(E)	

Los estados B y C son equivalentes.

Figura 9.3 Equivalencia de estados mediante partición.

EJEMPLO 9.2

Utilizar el método de partición para reducir la tabla de estados de la figura 9.4a.

Las particiones para la tabla de estados son

$$P_1 = (AD)(BE)(CF)(GH)$$
$$P_2 = (AD)(BE)(CF)(G)(H)$$
$$P_3 = P_2$$

La figura 9.4b muestra la tabla de estados reducida, con los siguientes estados simbólicos:

$$A' = (AD), \quad C' = (CF), \quad E' = (H),$$
$$B' = (BE), \quad D' = (G)$$

	$x=0$	$x=1$
A	$E/0$	$D/0$
B	$A/1$	$F/0$
C	$C/0$	$A/1$
D	$B/0$	$A/0$
E	$D/1$	$C/0$
F	$C/0$	$D/1$
G	$H/1$	$G/1$
H	$C/1$	$B/1$

(a)

	$x=0$	$x=1$
A'	$B'/0$	$A'/0$
B'	$A'/1$	$C'/0$
C'	$C'/0$	$A'/1$
D'	$E'/1$	$D'/1$
E'	$C'/1$	$B'/1$

(b)

Figura 9.4 Ejemplo de partición. **(a)** Tabla de estados. **(b)** Tabla de estados reducida.

EJEMPLO 9.3

Reducir la tabla de estados de la figura 9.5a mediante el método de partición.

Las particiones son

$$P_1 = (ACG)(BDEH)(F)$$
$$P_2 = (A)(CG)(BH)(DE)(F), \qquad \text{de la columna } x = 0$$
$$= (A)(C)(G)(BH)(DE)(F), \qquad \text{de la columna } x = 1$$
$$P_3 = P_2$$

Utilizamos los siguientes estados simbólicos para obtener la tabla de estados reducida de la figura 9.5b.

$$A' = (A), \quad C' = (C), \quad E' = (BH),$$
$$B' = (F), \quad D' = (G), \quad F' = (DE)$$

EJEMPLO 9.4

Este ejemplo ilustra el hecho de que las técnicas descritas son aplicables a circuitos secuenciales con varias entradas. La figura 9.6a muestra la tabla de estados para un circuito secuencial con dos líneas de entrada.

Las particiones son

$$P_1 = (ADFG)(BCEH)$$
$$P_2 = (AFG)(D)(BCEH)$$
$$P_3 = (AF)(G)(D)(BCH)(E)$$
$$P_4 = P_3$$

	x = 0	x = 1
A	$A/0$	$B/0$
B	$H/1$	$C/0$
C	$E/0$	$B/0$
D	$C/1$	$D/0$
E	$C/1$	$E/0$
F	$F/1$	$G/1$
G	$B/0$	$F/0$
H	$H/1$	$C/0$

(a)

	x = 0	x = 1
A'	$A'/0$	$E'/0$
B'	$B'/1$	$D'/1$
C'	$F'/0$	$E'/0$
D'	$E'/0$	$B'/0$
E'	$E'/1$	$C'/0$
F'	$C'/1$	$F'/0$

(b)

Figura 9.5 Ejemplo de partición. **(a)** Tabla de estados. **(b)** Tabla de estados reducida.

	x_1x_2 = 00	01	11	10
A	$D/0$	$D/0$	$F/0$	$A/0$
B	$C/1$	$D/0$	$E/1$	$F/0$
C	$C/1$	$D/0$	$E/1$	$A/0$
D	$D/0$	$B/0$	$A/0$	$F/0$
E	$C/1$	$F/0$	$E/1$	$A/0$
F	$D/0$	$D/0$	$A/0$	$F/0$
G	$G/0$	$G/0$	$A/0$	$A/0$
H	$B/1$	$D/0$	$E/1$	$A/0$

(a)

	x_1x_2 = 00	01	11	10
A'	$C'/0$	$C'/0$	$A'/0$	$A'/0$
B'	$B'/1$	$C'/0$	$D'/1$	$A'/0$
C'	$C'/0$	$B'/0$	$A'/0$	$A'/0$
D'	$B'/1$	$A'/0$	$D'/1$	$A'/0$
E'	$E'/0$	$E'/0$	$A'/0$	$A'/0$

(b)

Figura 9.6 Ejemplo de entrada múltiple. **(a)** Circuito original. **(b)** Circuito reducido.

La figura 9.6b muestra la tabla de estados reducida, donde hemos utilizado la siguiente sustitución de estados:

$$A' = (AF), \quad C' = (D), \quad E' = (G)$$
$$B' = (BCH), \quad D' = (E),$$

9.2.3 Tabla de implicación

La tabla de implicación es otra herramienta que puede servir para determinar la equivalencia de estados. Esta técnica es más general, pues se puede aplicar a circuitos secuenciales con especificación incompleta; sin embargo, también puede ocupar mayor tiempo que el método de partición.

Consideremos nuevamente el ejemplo de la figura 9.1b, repetido en la figura 9.7a. Utilizaremos este ejemplo para explicar el procedimiento.

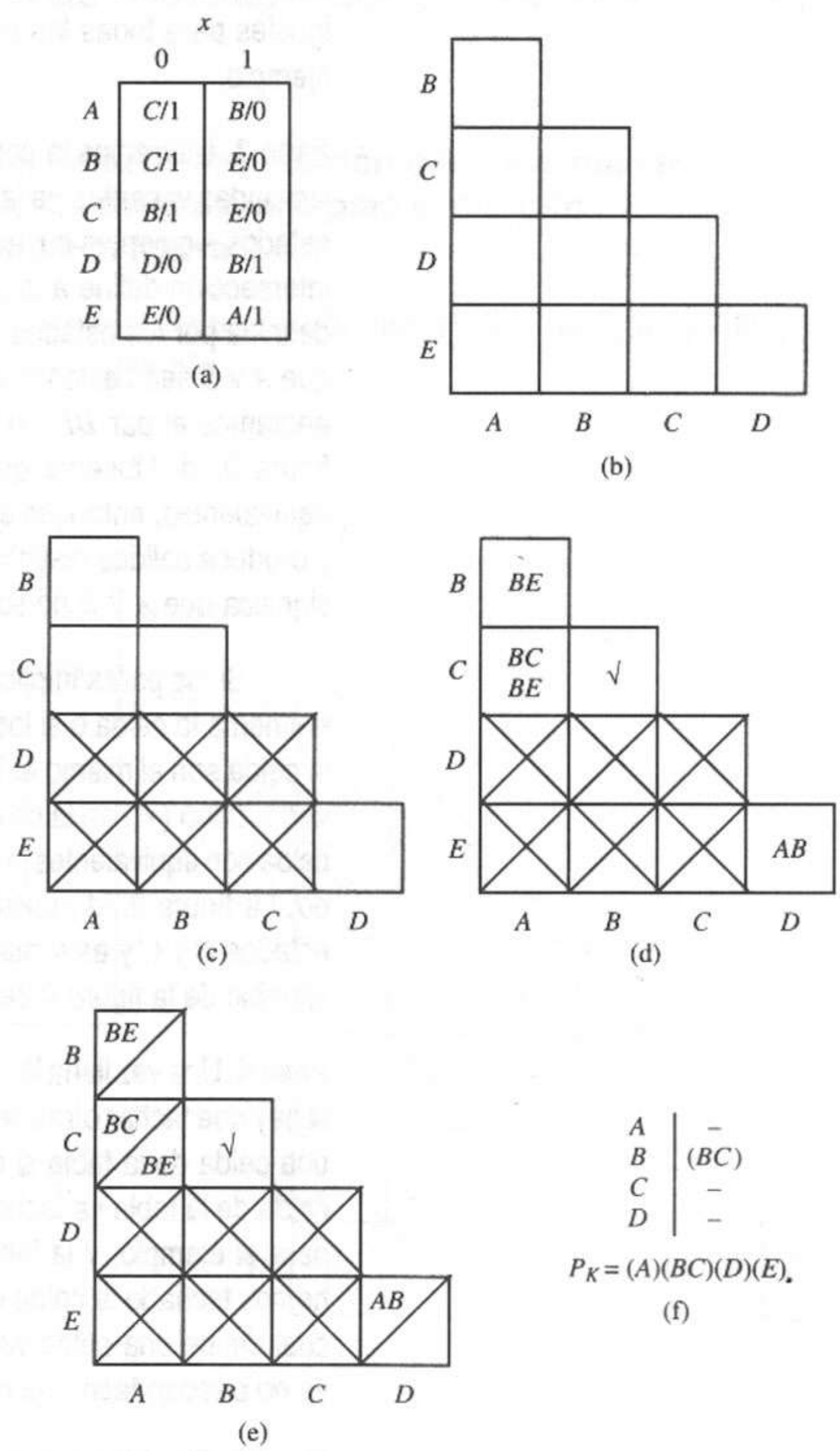


Figura 9.7 Tabla de implicación para un circuito de cinco estados. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Partición de salida. **(d)** Pares implicados. **(e)** Tabla terminada. **(f)** Partición de equivalencia.

Procedimiento de la tabla de implicación

Paso 1. Formamos una tabla mediante la estructura que se muestra en la figura 9.7b, que se obtiene listando en forma vertical todos los estados de la tabla, excepto el último, y anotando en sentido horizontal todos los estados de la tabla, excepto el primero. La tabla resultante muestra todas las combinaciones posibles de dos estados y, por tanto, cada celda de la tabla correspondiente a la intersección de una fila y una columna representa dos estados cuya equivalencia queremos verificar.

Paso 2. Puesto que sólo los estados con salidas idénticas podrían ser equivalentes (condición 1 para los estados equivalentes), colocamos una cruz en las celdas correspondientes a los pares de estados cuyas salidas no son iguales para todas las entradas. Hemos hecho esto en la figura 9.7c para el ejemplo.

Paso 3. Utilizamos la condición 2 de los estados equivalentes para completar las celdas vacantes de la figura 9.7c. Colocamos en estos bloques los pares de estados siguientes cuya equivalencia es "implicada" por los dos estados cuya intersección define a la celda. Como muestra de esto, consideremos la celda definida por los estados A y B. De la tabla de estados podemos ver que, para que A y B sean estados equivalentes, B y E deben ser equivalentes. Por tanto, anotamos el par BE en la celda definida por A y B, como se muestra en la figura 9.7d. Observe que si los estados del par implicado, B y E, no son equivalentes, entonces existe una cadena de entrada que comienza con $x = 0$ y produce salidas distintas dependiendo de si el estado inicial es A o B, lo que significa que A y B no son equivalentes.

Si los pares implicados para cada celda sólo contienen los estados que definen a la celda o si los estados siguientes de los dos estados que definen a la celda son el mismo estado para una entrada dada, colocamos una marca de verificación (✓) en la celda, lo cual indica que los dos estados que definen a la celda son equivalentes por *inspección* e independientes de cualquier par implicado. La figura 9.7d muestra esta condición mediante la celda definida por los estados B y C y es similar al caso de la equivalencia por inspección vista en el ejemplo de la figura 9.2e.

Paso 4. Una vez llena la tabla, se recorre ésta en forma sucesiva para determinar si hay que tachar otras celdas distintas de las tachadas en el paso 2. Tachamos una celda de la tabla si contiene al menos un par implicado que defina a una celda de la tabla ya tachada con anterioridad. Hemos realizado esta operación para el ejemplo, y la tabla resultante aparece en la figura 9.7e. Por ejemplo, hemos tachado la celda definida por A y B debido a que contiene el par BE, el cual define una celda ya eliminada. Repetimos este procedimiento hasta que ya no puedan tacharse más celdas.

Paso 5. Por último, obtenemos la tabla de la figura 9.7f enumerando en una columna los estados que definen la fila horizontal de la tabla de implicación. Después analizamos la tabla de implicación, columna por columna, de izquierda a derecha, para ver si no se han tachado otras celdas. Los estados que definen a las celdas no tachadas son equivalentes y se enumeran como pares equivalentes en la tabla de la figura 9.7f. Los pares se combinan mediante la transitividad.

$$(s_i, s_j)(s_j, s_k) \rightarrow (s_i, s_j, s_k) \tag{9.1}$$

En el ejemplo, todas las celdas de las columnas A, C y D se tachan y, por tanto, escribimos guiones en las filas correspondientes de la tabla de la figura 9.7f. En la columna B de la tabla de implicación, no tachamos la celda definida por los estados B y C y, por tanto, colocamos el par (BC) en la fila B de la tabla. Así, la partición de equivalencia consta de todos los estados equivalentes determinados

en la tabla; es decir, (BC) junto con los demás estados del circuito que no son equivalentes a algún otro estado. Observe que esta partición de equivalencia es idéntica a la obtenida anteriormente mediante el método de partición.

EJEMPLO 9.5

Utilizar una tabla de implicación para determinar la partición de equivalencia para el circuito secuencial de la figura 9.4a.

Repetimos el circuito como el de la figura 9.8a. Las figuras 9.8b y c muestran el análisis para este ejemplo.

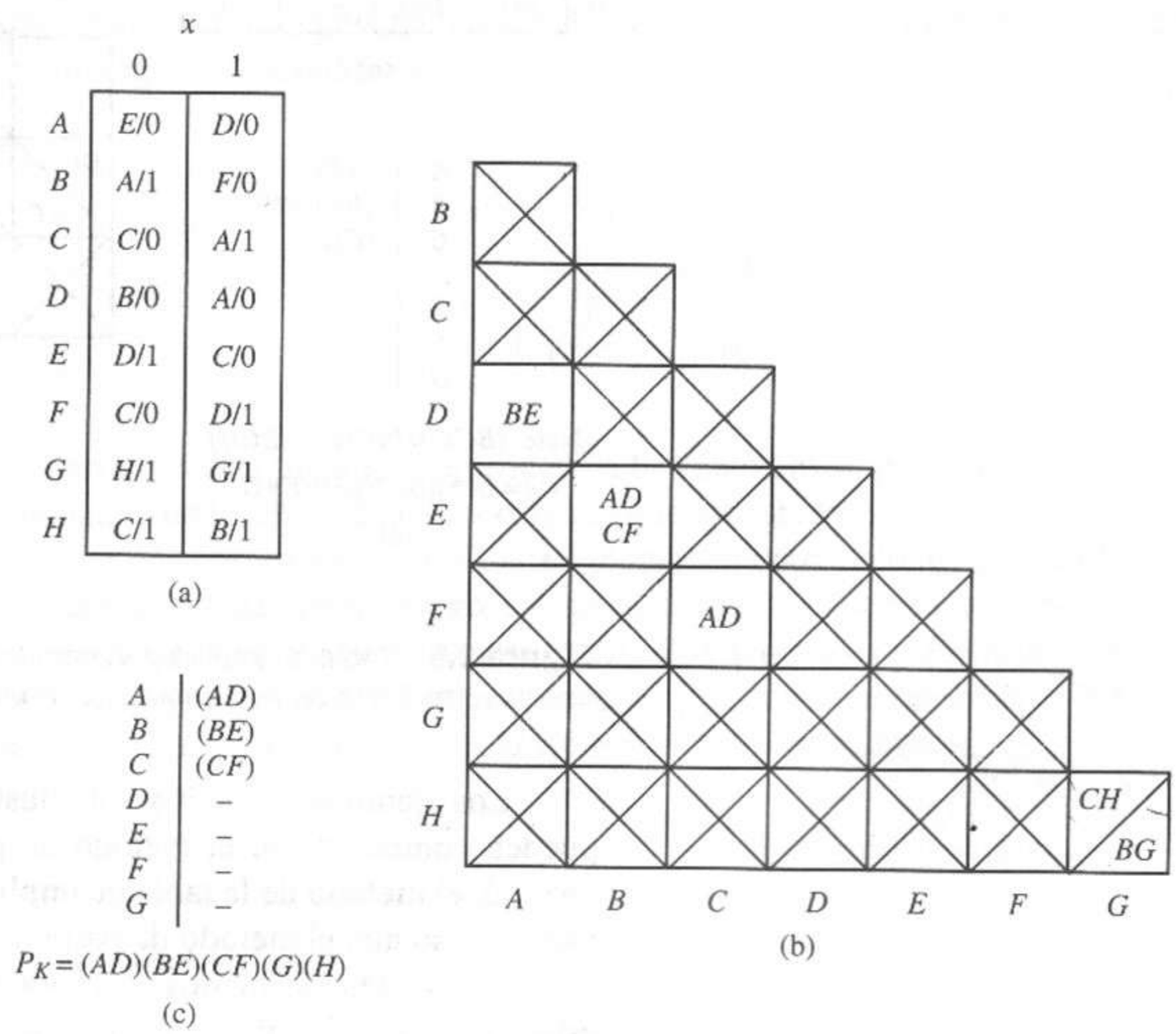


Figura 9.8 Tabla de implicación para un circuito de ocho estados. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Partición de equivalencia.

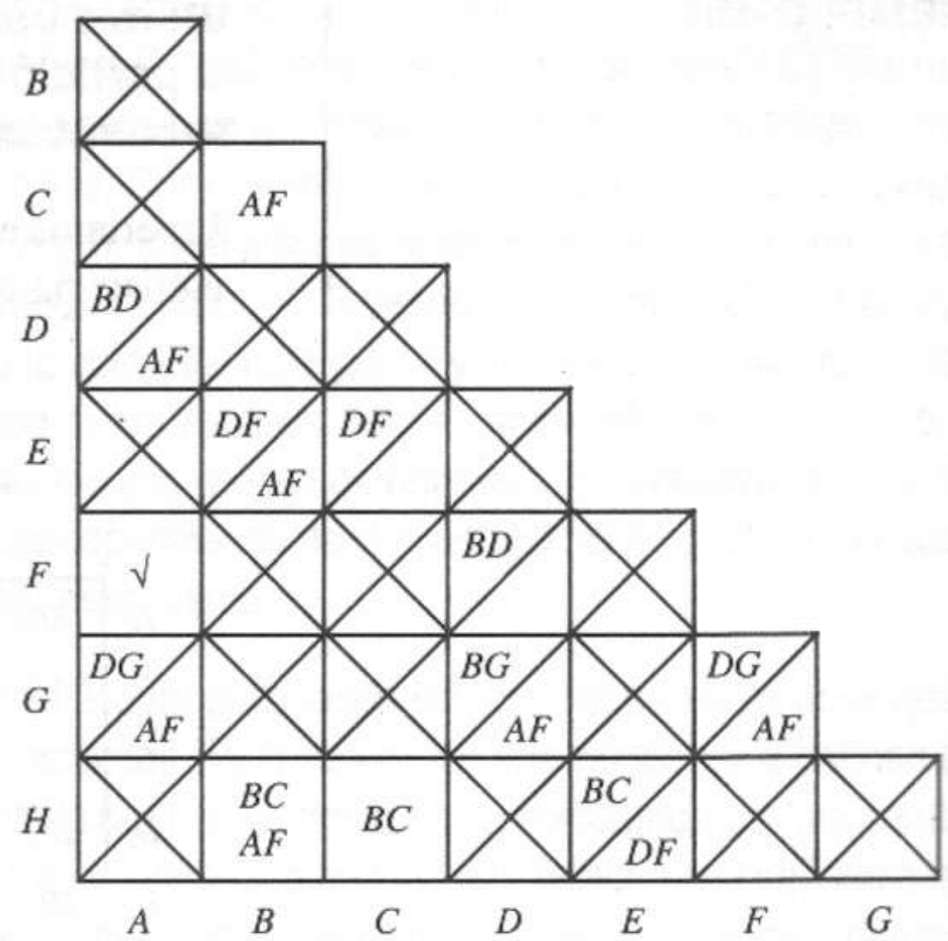
EJEMPLO 9.6

La partición de equivalencia para el circuito secuencial descrito por la tabla de estados de la figura 9.6a está dada en la figura 9.9.

Este ejemplo, aunque directo, tiene una característica sobresaliente. En la fila B de la figura 9.9c aparece el conjunto de estados equivalentes $(BC)(BH)$, mientras que (CH) aparece en la fila C. Podemos utilizar la ecuación 9.1 para combinar estos estados en una clase de equivalencia mayor, (BCH).

	x_1x_2 00	01	11	10
A	D/0	D/0	F/0	A/0
B	C/1	D/0	E/1	F/0
C	C/1	D/0	E/1	A/0
D	D/0	B/0	A/0	F/0
E	C/1	F/0	E/1	A/0
F	D/0	D/0	A/0	F/0
G	G/0	G/0	A/0	A/0
H	B/1	D/0	E/1	A/0

(a)



(b)

A	(AF)
B	(BC)(BH)
C	(CH)
D	–
E	–
F	–
G	–

Note: $(BC)(BH)(CH) = (BCH)$

$P_K = (AF)(BCH)(D)(E)(G)$

(c)

Figura 9.9 Tabla de implicación para un circuito de siete estados. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Partición de equivalencia.

Los ejemplos 9.5 y 9.6, que ilustran el uso de la tabla de implicación, se pueden comparar con el método de partición utilizado con anterioridad. En general, el método de la tabla de implicación es más rutinario, pero también es más tedioso que el método de partición.

En esta sección hemos analizado tres técnicas para minimizar el número de estados de los circuitos secuenciales síncronos. Se puede utilizar cualquiera de ellos en el proceso de síntesis.

9.3 Reducción de estados en circuitos con especificación incompleta

La minimización de las tablas de estados que contienen condiciones prescindibles requiere un análisis especial. Los siguientes ejemplos ilustran este punto.

EJEMPLO 9.7

Considere el problema de minimizar el siguiente circuito con especificación incompleta:

	x = 0	x = 1
A	$B/$—	$E/0$
B	$B/1$	$E/$—
C	$F/0$	$C/0$
D	$B/1$	$A/1$
E	$D/0$	$C/$—
F	$D/$—	$C/1$

En esta tabla de estados aparecen cuatro condiciones prescindibles, las cuales, por supuesto, pueden asignarse o especificarse de cualquier forma. Una vez especificadas estas condiciones prescindibles, la tabla de estados ya tiene una especificación completa, y podemos aplicar las técnicas de reducción de estado ya descritas para determinar los estados equivalentes y con ello reducir la tabla. Supongamos entonces que asignamos las condiciones prescindibles de modo que el estado A sea equivalente a B, y E sea equivalente a F. En estas condiciones, la tabla anterior se reduce a

	x = 0	x = 1
A'	$A'/1$	$D'/0$
B'	$D'/0$	$B'/0$
C'	$A'/1$	$A'/1$
D'	$C'/0$	$B'/1$

Observe que ésta fue una simplificación bastante obvia, pues podríamos ver de inmediato de la tabla de estados original que los estados A y B son equivalentes, al igual que los estados E y F, si asignamos las condiciones prescindibles de manera adecuada. Sin embargo, si especificamos las condiciones prescindibles como cero en la tabla original para los estados actuales A y E y como 1 para los estados B y F, entonces los estados A, C y E son equivalentes, al igual que los estados B, D y F. Por tanto, la tabla de estados se reduce a

	x = 0	x = 1
A'	$B'/0$	$A'/0$
B'	$B'/1$	$A'/1$

Observe que esta última simplificación no es evidente, y que proporciona la tabla más sencilla.

Las siguientes ideas forman la base para la reducción de la tabla de estados en el caso de circuitos con especificación incompleta.

9.3.1 Compatibilidad de estados

Debido a los problemas asociados con la asignación de valores a las condiciones prescindibles, debemos desarrollar un procedimiento distinto para eliminar los estados redundantes de las tablas de estados para circuitos con especificación incompleta. Comenzaremos con un análisis de las secuencias de entradas aplicables a un circuito con especificación incompleta; después definiremos la compatibilidad de estados y los métodos para la identificación de los estados compatibles e incompatibles.

Secuencias de entradas aplicables

Una secuencia de entrada es *aplicable* al estado S_i de una máquina con especificación incompleta si y sólo si se cumple la siguiente condición: Cuando la máquina está en S_i y se aplica la secuencia de entrada, todos los estados siguientes quedan especificados, excepto tal vez el último elemento de la secuencia. Por ejemplo, las secuencias de entrada 0111 y 1111 son aplicables al estado A de la máquina definida en la figura 8.49, pero la secuencia 11111 no es aplicable al estado A.

Estados compatibles

Dos estados S_i y S_j de una máquina con especificación incompleta son *compatibles* (S_iS_j) si y sólo si para cada secuencia de entrada aplicable a S_i y S_j se obtiene la misma secuencia de entrada cuando se especifican las salidas, sin importar cuál de los estados S_i y S_j sea el inicial. Los estados A y C en el ejemplo de reducción de estados son compatibles. Observe que las siguientes secuencias de salida producidas por la secuencia de entrada 1111 para los estados iniciales A y C son iguales cuando quedan especificadas:

Entrada:		1		1		1		1	
Estado:	A		E		C		C		C
Salida:		0		–		0		0	
Estado:	C		C		C		C		C
Salida:		0		0		0		0	

Se puede demostrar que la compatibilidad de estados define una relación de compatibilidad en el conjunto de estados de una máquina con especificación incompleta. Por tanto, un conjunto de estados compatibles es una *clase de compatibilidad.* Un *compatible maximal* es una clase de compatibilidad que deja de ser una clase de compatibilidad si se agrega un estado que no esté en la clase. En el ejemplo anterior, (AC), (AE), (CE) y (ACE) son clases de compatibilidad. De estas cuatro clases de compatibilidad, sólo (ACE) es compatible maximal.

Los estados S_i y S_j de una máquina con especificación incompleta son compatibles si y sólo si se cumplen las dos condiciones siguientes:

1. Las salidas producidas por S_i y S_j deben ser iguales, cuando ambos se especifican, para cada entrada posible I_p.
2. Los estados siguientes de S_i y S_j deben ser compatibles cuando ambos se especifican, para cada entrada posible I_p.

Estados incompatibles

Dos estados de una máquina con especificación incompleta que no satisfacen las dos condiciones anteriores son *incompatibles*. Un conjunto de estados incompatibles forma una *clase de incompatibilidad.* Un *incompatible maximal* es una clase de incompatibilidad a la que no puede agregarse otro estado incompatible sin destruir la clase.

EJEMPLO 9.8

Determinar las clases de compatibilidad y los compatibles maximales para la tabla de estados de la figura 9.10a.

Primero utilizamos la tabla de implicación para determinar todos los pares de estados compatibles, como se muestra en la figura 9.10b. Formamos esta tabla como en el caso con especificación completa. Observe que la presencia de términos de salida prescindibles permite aparear el estado *B* con el estado *A* en una instancia y el estado *C* en otra.

También realizamos la reducción de la tabla de implicación del mismo modo que en el caso con especificación completa. La figura 9.10c muestra la determinación de las clases de compatibilidad. Observe que este procedimiento recuerda al utilizado para determinar los estados equivalentes en el caso con especificación completa. En la fila definida por el estado *G*, el par (*GH*) es una clase de compatibilidad. Este par se toma de la última columna de la tabla de implicación. A continuación agregamos los pares compatibles de la columna *F* de la tabla de implicación. La fila *F* de la lista de la figura 9.10c contiene ahora dos pares compatibles, (*GH*)(*FG*). Al agregar más columnas, yendo de derecha a izquierda en la tabla de implicación, la lista de pares compatibles crece. Por ejemplo, la fila *E* contiene los pares (*EG*)(*EH*)(*GH*)(*FG*); pero (*EG*)(*EH*)(*GH*) se pueden fusionar en una clase de compatibilidad mayor (*EGH*), de modo que incluimos una segunda fila para el estado *E* en la figura 9.10c que muestra este paso. Continuamos el proceso de agregación de los pares compatibles y los combinamos en clases de compatibilidad más grandes al movernos de derecha a izquierda en la tabla de implicación. La fila inferior de la figura 9.10c es el conjunto de compatibles maximales.

Un análisis de la lista de compatibles maximales ilustra la diferencia fundamental entre la compatibilidad en los circuitos con especificación incompleta y la equivalencia en los circuitos con especificación completa. En un circuito con especificación completa, si determinamos que el estado *A* fue equivalente al estado *B* y que el estado *B* es equivalente al estado *C*, tenemos garantizado sin verificarlo que los estados *A* y *C* son equivalentes, pues la equivalencia de estados es una relación de equivalencia. En cambio, como la propiedad transitiva no se cumple en general para los circuitos con especificación incompleta, no existe tal garantía. Consideremos el compatible maximal (*BCG*) de la figura 9.10c. Para poder agrupar los tres estados debemos tener los pares compatibles (*BC*),(*CG*) y (*BG*); es decir, (*BC*) y (*CG*) no implican en forma automática a (*BG*). Un poco de reflexión nos mostrará que este problema surge debido a las condiciones prescindibles de la tabla de estados.

EJEMPLO 9.9

Determinar las clases de incompatibilidad y los incompatibles maximales para la máquina secuencial de la figura 9.10a.

Utilizamos la tabla de implicación de la figura 9.10b para generar las clases de incompatibilidad, extrayendo de ellas pares de estados que *no* son compatibles, como muestra la figura 9.10d. El conjunto de incompatibles maximales es la lista de la parte inferior de la tabla.

	x = 0	x = 1
A	A/–	C/1
B	B/–	A/–
C	G/–	E/0
D	C/1	C/–
E	A/1	C/–
F	D/–	A/–
G	G/–	G/–
H	H/–	D/–

(a)

B	AC					
C		BG AE				
D	AC	BC / AC	CG CE			
E	√	AB / AC	AG	AC		
F	AD / AC	BD	DG AE	CD / AC	AD / AC	
G	CG	AG	EG	CG	AG CG	DG AG
H	CD	AD	GH / DE	CH / CD	AH CD	DH / AD	DG
	A	B	C	D	E	F	G

(b)

G	(GH)
F	(GH)(FG)
E	(EG)(EH)(GH)(FG)
E	(FG)(EGH)
D	(DG)(FG)(EGH)
C	(CG)(CF)(CE)(CD)(DG)(FG)(EGH)
C	(CEG)(CDG)(CFG)(EGH)
B	(BC)(BG)(CEG)(CDG)(CFG)(EGH)
A	(AE)(AG)(AH)(BC)(BG)(CEG)(CDG)(CFG)(EGH)
A	(AEG)(AGH)(AEH)(BCG)(CEG)(CDG)(CFG)(EGH)
A	(AEGH)(BCG)(CDG)(CEG)(CFG)

(c)

G	–
F	(FH)
E	(FH)(EF)
D	(FH)(EF)(DH)(DF)(DE)
D	(FH)(DH)(DEF)
C	(CH)(FH)(DH)(DEF)
B	(BH)(BF)(BE)(BD)(CH)(FH)(DH)(DEF)
B	(BH)(BDEF)(CH)(FH)(DH)
B	(BDEF)(CH)(BDFH)
A	(AB)(AC)(AD)(AF)(BDEF)(CH)(BDFH)
A	(ABDF)(AC)(BDEF)(CH)(BDFH)

(d)

Figura 9.10 Generación de compatibles e incompatibles maximales. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Clases de compatibilidad. **(d)** Clases de incompatibilidad.

El proceso de generación de los compatibles e incompatibles maximales demostrado en las figuras 9.10c y d puede ser algo tedioso, por lo que presentaremos una técnica gráfica que nos ayudará en este proceso.

Diagramas de fusión

El proceso de determinación de los conjuntos compatibles maximales de estados a partir de los pares compatibles obtenidos a partir de la tabla de implicación es apoyado en gran medida por una técnica gráfica, el *diagrama de fusión*. En primer lugar, representamos de manera conveniente los estados de la máquina original como puntos con una distribución uniforme en torno de un círculo; después utilizamos una línea para unir cada par relacionado (compatible) de estados. Esto concluye la construcción del diagrama de fusión. Podemos hacer lo mismo para determinar conjuntos incompatibles maximales de estados.

Los conjuntos maximales de estados pueden deducirse a partir del diagrama de fusión, buscando los conjuntos en que cada estado está unido a cada uno de los demás estados mediante un segmento de recta. Así, los conjuntos maximales forman patrones gráficos regulares, como muestra la figura 9.11. Las reglas para la extracción de conjuntos maximales a partir de un diagrama de fusión son las siguientes:

Regla 1. Hacer lo más grande posible cada conjunto maximal.

Regla 2. Cada estado del conjunto maximal debe estar interconectado con los demás estados del conjunto mediante un segmento de recta.

Regla 3. Cada par de estados relacionados entre sí (compatible o incompatible) debe aparecer en al menos un conjunto maximal.

Mostraremos la aplicación de estas reglas mediante un ejemplo.

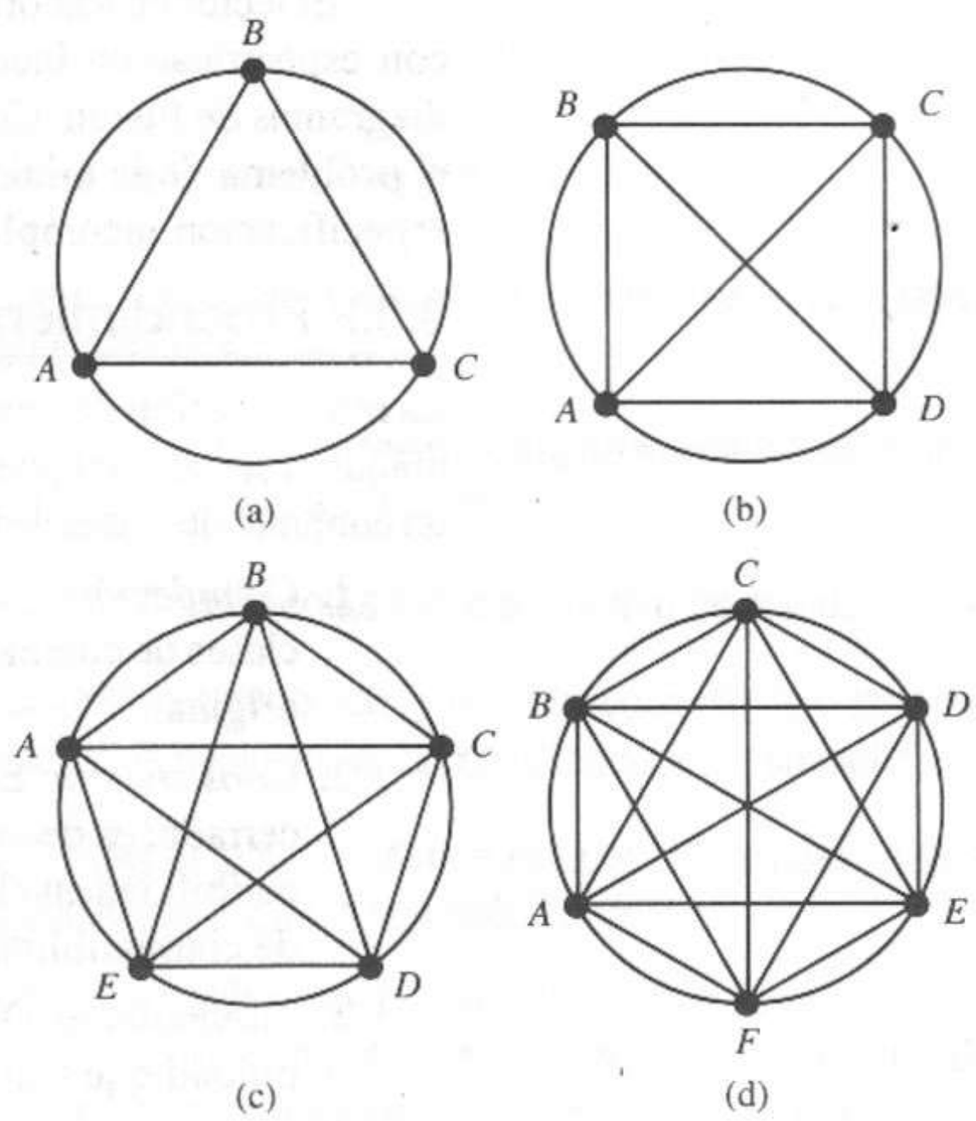


Figura 9.11 El diagrama de fusión. **(a)** Tres estados. **(b)** Cuatro estados. **(c)** Cinco estados. **(d)** Seis estados.

EJEMPLO 9.10

Hemos construido los diagramas de fusión de la figura 9.12 a partir de la tabla de implicación de la figura 9.10b. Consideramos el diagrama de fusión para los compatibles maximales en la figura 9.12a. Hemos enfatizado el patrón gráfico para el compatible maximal (*AEGH*). Observe que cualquier intento por agregar otro estado compatible a esta agrupación gráfica falla. Los demás segmentos de recta (pares compatibles) se pueden cubrir mediante cuatro triángulos (*BCG*),(*CDG*),(*CEG*) y (*CFG*). Por tanto, hemos determinado el conjunto de compatibles maximales. Obtenemos el conjunto de incompatibles maximales a partir de la figura 9.12b de manera similar.

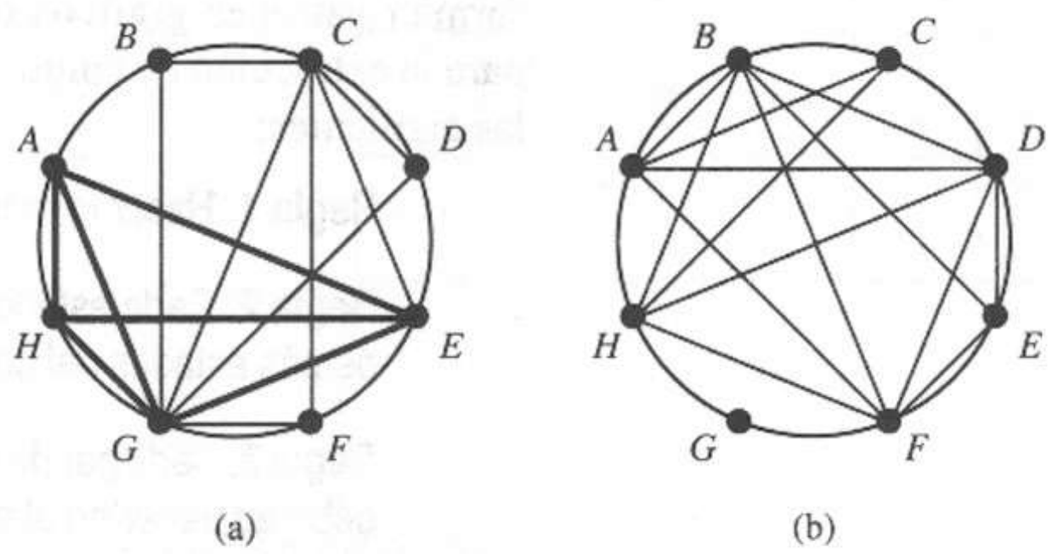


Figura 9.12 Ejemplos de diagramas de fusión. **(a)** Diagrama de fusión para los compatibles maximales. **(b)** Diagrama de fusión para los incompatibles maximales.

El lector está ahora familiarizado con el concepto de circuitos secuenciales con especificación incompleta, estados compatibles, estados incompatibles y diagramas de fusión. Con estas herramientas estamos preparados para enfrentar el problema de la minimización de estados para los circuitos secuenciales con especificación incompleta.

9.3.2 Procedimiento de minimización

La minimización de una tabla de estados con especificación incompleta para una máquina secuencial puede ser un proceso complejo. En general, debemos elegir un conjunto de clases de compatibilidad que cumpla las tres condiciones siguientes:

1. *Compleción*: La unión de todos los conjuntos en el conjunto elegido de clases de compatibilidad debe contener a todos los estados de la máquina original.
2. *Consistencia*: El conjunto elegido de clases de compatibilidad debe ser cerrado; es decir, los estados siguientes implicados de cada clase de compatibilidad en el conjunto elegido deben estar contenidos en alguna clase de compatibilidad dentro del conjunto.
3. *Minimalidad*: Debe elegirse el número más pequeño de clases de compatibilidad que cumpla con el criterio anterior.

Una vez determinado un conjunto de clases de compatibilidad que satisfaga estas condiciones, cada clase del conjunto corresponde a un estado de la tabla de estados reducida. Por desgracia, el proceso de selección del conjunto de clases

de compatibilidad que cumplen las tres condiciones debe realizarse mediante prueba y error. Por tanto, sería útil acotar al menos el número K de estados necesarios para realizar un circuito minimal.

La cota superior U sobre el número de estados del circuito minimal está dada por la expresión

$$U = \text{mínimo}\{\text{NSMC},\text{NSOC}\}$$

donde

NSMC = número de conjuntos de compatibles maximales

NSOC = número de estados en el circuito original

Esta ecuación sólo establece que no necesitamos más estados en el circuito minimal que el número de estados en el circuito original, y que si existe alguna compatibilidad entre estados tal que NSMC < NSOC, entonces necesitaremos menos de NSOC estados.

La cota inferior L sobre el número de estados en el circuito minimal está dada por la expresión

$$L = \text{máximo}\{\text{NSMI}_1,\text{NSMI}_2,\ldots,\text{NSMI}_p,\ldots\}$$

donde

NSMI_i = número de estados en el i-ésimo grupo del conjunto de maximales incompatibles del circuito original

Esta condición es razonable, pues si existen dos estados incompatibles en el circuito original, el circuito minimal deberá tener al menos dos estados para distinguir los incompatibles.

Ahora podemos especificar el algoritmo para la reducción de estados en las máquinas secuenciales con especificación incompleta.

Algoritmo para la reducción de estados

Paso 1. Determinar los compatibles maximales mediante la tabla de implicación y el diagrama de fusión.

Paso 2. Determinar los maximales incompatibles mediante la tabla de implicación del paso 1 y otro diagrama de fusión.

Paso 3. Determinar las cotas sobre el número de estados necesarios. U y L.

Paso 4. Determinar, por prueba y error, un conjunto de clases de compatibilidad que satisfagan la compleción, la consistencia y la minimalidad.

Paso 5. Obtener la tabla mínima de estados. En general, aún puede contener estados siguientes y salidas no especificados.

Podemos iniciar la selección por prueba y error de las clases de compatibilidad considerando sólo los compatibles maximales. El conjunto de compatibles maximales es *completo* y *consistente*. Sin embargo, el conjunto podría no ser minimal. Podemos iniciar la búsqueda de un conjunto minimal de clases de compatibilidad considerando los compatibles maximales en grupos de L, la cota inferior.

EJEMPLO 9.11

Determinar la tabla de estados reducida para la máquina con especificación incompleta de la figura 9.10a.

Primero, construimos una *tabla cerrada*, considerando a los compatibles maximales como estados y determinando sus conjuntos de estados siguientes, como en la figura 9.13a. Obtenemos cada entrada de la tabla a partir de la tabla original, registrando el estado siguiente de cada estado dentro de un compatible maximal.

	$x=0$	$x=1$
$(AEGH)$	AGH	CDG
(BCG)	BG	AEG
(CDG)	CG	CEG
(CFG)	DG	AEG
(CEG)	AG	CEG

(a)

	$x=0$	$x=1$
A'	$A'/1$	$C'/1$
B'	$B'/-$	$A'/0$
C'	$B', C', D', E'/1$	$D'/0$
D'	$A'/1$	$D'/0$
E'	$C'/-$	$A'/0$

(b)

Figura 9.13 Reducción de estados. **(a)** Tabla cerrada. **(b)** Tabla de estados reducida.

Como NSMC = 5 y NSOC = 8, la cota superior sobre el número de estados es

$$U = \text{mín}\{5,8\} = 5$$

Determinamos la cota inferior mediante los maximales incompatibles. En la figura 9.12b hemos deducido el conjunto de maximales incompatibles, que es

$$(ABDF)(BDEF)(BDFH)(AC)(CH)$$

Por tanto,

$$L = \text{máx}\{4,4,4,2,2\} = 4$$

El número de estados en la máquina reducida está acotado por

$$4 \le K \le 5$$

Como queremos obtener un circuito minimal, comenzamos con la cota inferior para ver si podemos determinar cuatro compatibles maximales que satisfagan las condiciones de compleción y consistencia. Con un proceso de prueba y error determinamos, mediante la tabla cerrada de la figura 9.13a, que ningún conjunto de cuatro compatibles maximales satisface la compleción y la consistencia. Por tanto, necesitamos cinco compatibles maximales. Por definición, el conjunto de compatibles maximales es completo y consistente, así que la máquina reducida tendrá los cinco estados

$$A' = (AEGH), \quad D' = (CEG)$$
$$B' = (BCG), \quad E' = (CFG)$$
$$C' = (CDG)$$

La figura 9.13b muestra la tabla de estados reducida que duplica la función de la figura 9.10a.

Explicaremos ahora la entrada múltiple del estado siguiente en la figura 9.13b. Observe que si el circuito se encuentra en el estado C' y $x = 0$, entonces el estado siguiente puede ser B', C', D' o E'. Esta entrada múltiple del estado siguiente existe porque la tabla cerrada indica que, para $x = 0$, $(CDG) \rightarrow CG$, por lo que el estado siguiente puede ser (BCG), (CDG), (CEG) o (CFG).

En el ejemplo anterior utilizamos todos los compatibles maximales como estados de la máquina reducida. En algunos casos, sólo se puede elegir un subconjunto de compatibles maximales para la reducción de la máquina, como lo ilustra el siguiente ejemplo.

EJEMPLO 9.12

Determinar la tabla de estados reducida correspondiente a la dada en la figura 9.14a.

Realizamos los pasos 1 y 2 del algoritmo de reducción de estados en las figuras 9.14b, c y d. De la tabla de implicación, los pares compatibles son

$$(AB)(AC)(AD)(AE)(BD)(CD)(CE)$$

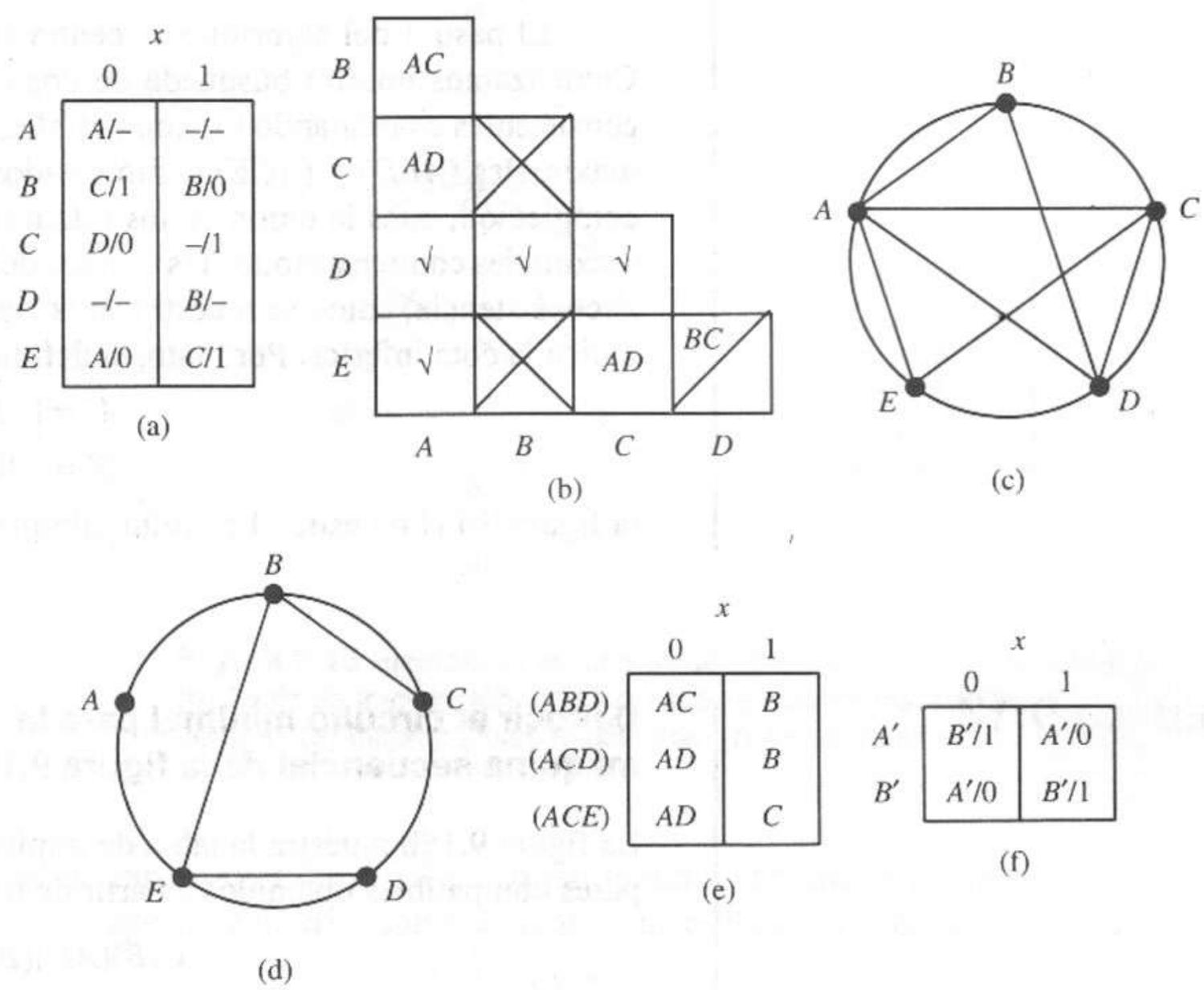


Figura 9.14 Ejemplo de reducción de estados. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Compatibles maximales. **(d)** Incompatibles maximales. **(e)** Tabla cerrada. **(f)** Tabla de estados reducida.

Procesamos estos pares compatibles en el diagrama de fusión de la figura 9.14c para obtener los compatibles maximales

$$(ABD)(ACD)(ACE)$$

De manera similar, determinamos los pares incompatibles a partir de la tabla de implicación como

$$(BC)(BE)(DE)$$

El diagrama de fusión de la figura 9.14d muestra que estos pares son los incompatibles maximales.

El paso 3 del algoritmo de reducción calcula las cotas sobre el número de estados reducidos. Los cálculos son:

$$\begin{aligned} \text{NSMC} &= 3 \\ \text{NSOC} &= 5 \\ \text{NSMI}_1 &= 2 \\ \text{NSMI}_2 &= 2 \\ \text{NSMI}_3 &= 2 \end{aligned}$$

Por tanto,

$$\begin{aligned} U &= \text{mín}\{3,5\} = 3 \\ L &= \text{máx}\{2,2,2\} = 2 \end{aligned}$$

y

$$2 \le K \le 3$$

El paso 4 del algoritmo se centra en la tabla cerrada de la figura 9.14e. Comenzamos nuestra búsqueda de dos clases de compatibilidad completas y consistentes examinando los compatibles maximales. Al elegir los compatibles maximales (ABD) y (ACE) como estados de prueba, vemos que satisfacen la compleción, pues la unión de los estados contenidos en estos dos compatibles maximales contiene a todos los estados del circuito original. También satisfacen la consistencia, como se muestra en la figura 9.14e, y la minimalidad, como lo indica la cota inferior. Por tanto, si definimos

$$\begin{aligned} A' &= (ABD) \\ B' &= (ACE) \end{aligned}$$

la figura 9.14f muestra el circuito minimal final para el paso 5 del algoritmo.

EJEMPLO 9.13

Deducir el circuito minimal para la máquina secuencial de la figura 9.15a.

La figura 9.15b muestra la tabla de implicación para esta tabla de estados. Los pares compatibles obtenidos a partir de la tabla son

$$(AB)(AD)(BC)(BD)$$

La figura 9.15c es el diagrama de fusión correspondiente, y los compatibles maximales obtenidos a partir del diagrama son

$$(ABD)(BC)(E)(F)$$

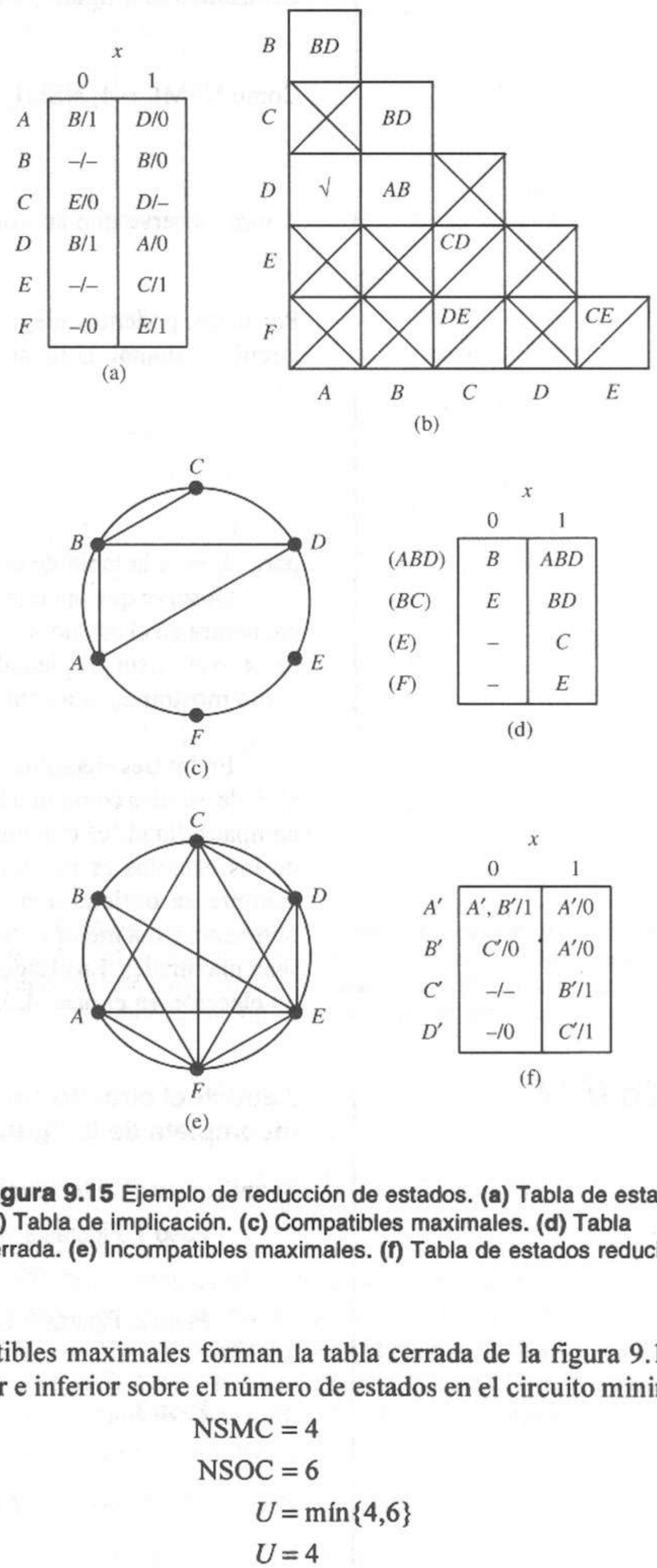


Figura 9.15 Ejemplo de reducción de estados. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Compatibles maximales. **(d)** Tabla cerrada. **(e)** Incompatibles maximales. **(f)** Tabla de estados reducida.

Estos compatibles maximales forman la tabla cerrada de la figura 9.15d. Las cotas superior e inferior sobre el número de estados en el circuito minimal son:

$$\text{NSMC} = 4$$

$$\text{NSOC} = 6$$

$$U = \text{mín}\{4,6\}$$

$$U = 4$$

Los pares incompatibles son

$$(AC)(AE)(AF)(BE)(BF)(CD)(CE)(CF)(DE)(DF)(EF)$$

La figura 9.15e muestra un diagrama de fusión para los maximales incompatibles. Utilizamos esta figura para deducir los maximales incompatibles, que son

$$(CDEF)(ACEF)(BEF)$$

Como $\text{NSMI}_1 = 4$, $\text{NSMI}_2 = 4$ y $\text{NSMI}_3 = 3$, entonces

$$L = \text{máx}\{4,4,3\}$$
$$L = 4$$

Ahora, observe que las cotas especifican que

$$4 \leq K \leq 4$$

Por tanto, podemos elegir todos los compatibles maximales como estados del circuito minimal. Utilizamos los símbolos

$$A' = (ABD)$$
$$B' = (BC)$$
$$C' = (E)$$
$$D' = (F)$$

para obtener la tabla de estados minimal en la figura 9.15f.

Observe que la tabla de estados resultante es muy flexible. Si el circuito se encuentra en el estado A' y aplicamos la entrada $x = 0$, el estado siguiente puede ser A' o B'. Esta propiedad será útil para simplificar la realización en hardware, como mostraremos a continuación.

En los tres ejemplos anteriores, enfocamos el paso 4 del algoritmo de reducción de estados como una búsqueda en un conjunto muy restringido de clases de compatibilidad, el conjunto de compatibles maximales. Aunque en términos de los cálculos es recomendable considerar sólo compatibles maximales, no siempre se obtiene una tabla de estados reducida minimal. Por lo general obtenemos una mejor reducción si no pedimos que las clases de compatibilidad sean maximales. En el siguiente ejemplo mostraremos un algoritmo para realizar tal elección en el paso 4, la cual siempre lleva a una tabla reducida minimal.

EJEMPLO 9.14

Reducir el circuito secuencial con especificación incompleta de la figura 9.16a.

Si aplicamos el algoritmo de reducción de estados, determinamos lo siguiente:

Paso 1. Figuras 9.16b y c. Los compatibles maximales son

$$(ABC)(ACD)(ADE)$$

Paso 2. Figuras 9.16b y c. Los incompatibles maximales son

$$(BD)(BE)(CE)$$

Paso 3.

$$\text{NSMC} = 3$$
$$\text{NSOC} = 5$$
$$\text{NSMI}_1 = 2$$
$$\text{NSMI}_2 = 2$$

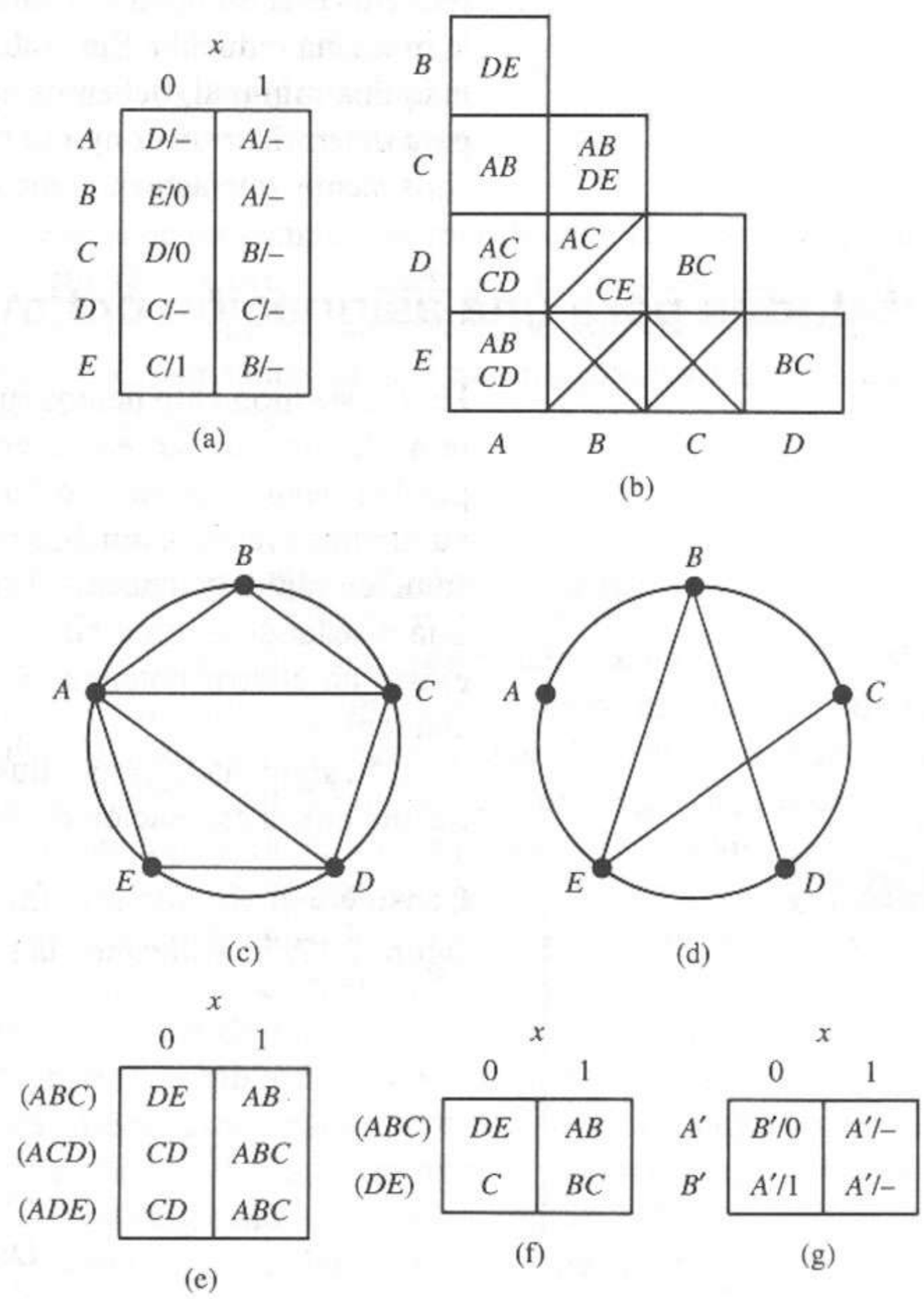


Figura 9.16 Un contraejemplo. **(a)** Tabla de estados. **(b)** Tabla de implicación. **(c)** Compatibles maximales. **(d)** Incompatibles maximales. **(e)** Tabla cerrada para compatibles maximales. **(f)** Tabla cerrada para incompatibles maximales. **(g)** Tabla de estados reducida.

$$\mathrm{NSMI}_3 = 2$$
$$U = \min\{3, 5\} = 3$$
$$L = \max\{2, 2, 2\} = 2$$
$$2 \leq K \leq 3$$

Paso 4. Consideremos las tablas cerradas de las figuras 9.16e y f. No podemos determinar un conjunto de dos compatibles maximales completo y consistente. Por tanto, si nos restringimos a las clases maximales de compatibilidad, deberemos elegir tres estados en la máquina reducida. Sin embargo, si elegimos las clases de compatibilidad (ABC) y (DE), determinamos que este conjunto es completo, consistente y minimal.

Paso 5. $A' = (ABC)$, $B' = (DE)$.

La figura 9.16g muestra la tabla de estados reducida minimal.

Esto completa nuestro análisis relativo a la reducción de estados de circuitos secuenciales con especificación incompleta. En general, obtenemos una máquina reducida casi minimal utilizando los compatibles maximales como estados para la máquina reducida. Sin embargo, si queremos garantizar la existencia de una máquina minimal, debemos aplicar el algoritmo desarrollado en este capítulo para determinar un conjunto minimal de clases de compatibilidad, completo y consistente, que actúen como estados para la máquina reducida.

9.4 Métodos para una asignación óptima de estados

Hasta este momento hemos supuesto alguna asignación de estados, sin analizar otra alternativa. Sin embargo, cabe señalar que dos asignaciones diferentes pueden llevar a grandes diferencias en el hardware. Desde un punto de vista puramente práctico, muchos ingenieros podrían argumentar que se necesita una solución rápida, no necesariamente minimal. Sin embargo, si el sistema que se está diseñando es un módulo básico que será fabricado en grandes cantidades, existe un ahorro potencial en el costo si el módulo se construye con pocos elementos.

El siguiente ejemplo ilustra la necesidad de ciertos criterios en la elección de una buena asignación de estados.

EJEMPLO 9.15

Considere el circuito secuencial descrito por la tabla de estados mínima de la figura 9.17. Si utilizamos la siguiente asignación de estados

	y_1	y_2	y_3
A:	0	0	0
B:	0	0	1
C:	0	1	1
D:	0	1	0
E:	1	0	1
F:	1	1	0
G:	1	1	1

entonces las ecuaciones lógicas que implantan el circuito son

$$
\begin{aligned}
J_1 &= \bar{y}_2 x + y_3 x, & J_3 &= \bar{y}_2 \\
K_1 &= \bar{y}_3 + x, & K_3 &= y_2 \\
J_2 &= y_3, & z &= \bar{y}_3 y_2 \bar{y}_1 \bar{x} + \bar{y}_3 y_1 x \\
K_2 &= \bar{y}_3
\end{aligned}
$$

Considere también la siguiente asignación:

	y_1	y_2	y_3
A:	0	0	0
B:	0	0	1
C:	0	1	0
D:	0	1	1
E:	1	0	0
F:	1	0	1
G:	1	1	0

Estado actual	x = 0	x = 1
A	$B/0$	$E/0$
B	$C/0$	$G/0$
C	$D/0$	$F/0$
D	$A/1$	$A/0$
E	$G/0$	$C/0$
F	$A/0$	$A/1$
G	$F/0$	$D/0$

Estado siguiente/salida

Figura 9.17 Una máquina de siete estados.

Con esta segunda asignación, las ecuaciones lógicas que implantan el circuito son

$$J_1 = x\bar{y}_3 + x\bar{y}_2, \qquad J_3 = y_2 + \bar{x}\bar{y}_1$$
$$K_1 = x + y_3, \qquad K_3 = 1$$
$$J_2 = y_1\bar{y}_3 + \bar{y}_1 y_3 \qquad z = xy_1y_3 + \bar{x}y_2y_3$$
$$K_2 = y_3 + \bar{x}y_1 + x\bar{y}_1$$

Un rápido conteo de las compuertas muestra que la primera asignación requiere tres compuertas OR, cuatro compuertas AND y una compuerta NOT; la segunda asignación requiere seis compuertas OR, nueve compuertas AND y una compuerta NOT. Por tanto, la segunda asignación requiere *dos veces* más compuertas que la primera.

9.4.1 Asignaciones de estados únicas

El número de posibles asignaciones de estados en un problema arbitrario puede ser grande. Por ejemplo, el número posibles de asignaciones de estados para una máquina de cinco estados es mayor que 100 y en el caso de una máquina de diez estados es mayor que 10 millones. Por desgracia, no existe una técnica sencilla y eficiente para elegir una asignación de estados [4–8]. En vez de una solución óptima, ofrecemos en las siguientes páginas algunos criterios para ayudar al lector a elegir una asignación de estados razonablemente buena.

Con esta motivación, examinaremos ahora diversas técnicas útiles para elegir una asignación de estados. Como parte del procedimiento de síntesis, hemos obtenido una tabla de estados mínima a fin de reducir el número de elementos de memoria necesarios para implantar el circuito secuencial síncrono. Una vez determinado el número mínimo de elementos de memoria, la elección adecuada de una asignación de estados reduce en forma drástica el número de compuertas lógicas necesarias para implantar las funciones de conmutación de excitación y de salida. Ya hemos visto que el número total de elementos de memoria N_{FF} se relaciona con el número de estados N_s del circuito mediante

$$2^{N_{FF}-1} < N_S \leq 2^{N_{FF}}$$

Por tanto, existirán

$$N_{SA} = \frac{2^{N_{FF}}!}{(2^{N_{FF}} - N_S)!}$$

formas de asignar las $2^{N_{FF}}$ combinaciones de asignaciones binarias de estados a los N_s estados. En la tabla 9.1 evaluamos esta expresión para diversos casos.

Mostraremos que no todas estas asignaciones son únicas con respecto de las ecuaciones de excitación y salida resultantes. Haremos esto en el ejemplo siguiente.

EJEMPLO 9.16

Para la tabla de estados de la figura 9.18a, comparar las ecuaciones de excitación del *flip–flop* D que resultan de una asignación arbitraria de variables de estados y_1 y y_2 a las asignaciones obtenidas al complementar y_1 e intercambiar y_1 y y_2.

TABLA 9.1 NÚMERO DE ASIGNACIONES DE ESTADO

N_S	N_{FF}	N_{SA}	N_{UA}
1	0	—	—
2	1	2	1
3	2	24	3
4	2	24	3
5	3	6,720	140
6	3	20,160	420
7	3	40,320	840
8	3	40,320	840
9	4	4.15×10^9	10,810,800
10	4	2.91×10^{10}	75,675,600

Estado actual	x = 0	x = 1
A	$A/0$	$B/0$
B	$A/0$	$C/0$
C	$C/0$	$D/0$
D	$C/1$	$A/0$

(a)

Estados	Asignaciones 1 $y_1\,y_2$	2 $y_1\,y_2$	3 $y_1\,y_2$
A	00	10	00
B	01	11	10
C	11	01	11
D	10	00	01

(b)

Figura 9.18 Asignaciones de estados equivalentes para un circuito secuencial. **(a)** Tabla de estados. **(b)** Asignaciones de estado.

Para la asignación 1 de la figura 9.18b, podemos deducir las siguientes ecuaciones de excitación.

$$D_1 = y_1\bar{x} + y_2x$$
$$D_2 = y_1\bar{x} + \bar{y}_1x$$

Podemos deducir otra asignación de estados complementando una posición de bit dada para obtener una nueva asignación de estados. Deducimos la asignación de estados 2 de la figura 9.18b de la asignación 1, complementando los valores de y_1. Las ecuaciones de excitación resultantes son

$$D_1 = y_1\bar{x} + \bar{y}_2x$$
$$D_2 = \bar{y}_1\bar{x} + y_1x$$

Podemos deducir estas ecuaciones a partir de las anteriores complementando y_1 y D_1. Las nuevas ecuaciones lógicas no son mejores que las anteriores, pues en la mayor parte de los dispositivos *flip–flop* están disponibles las variables verdaderas y complementadas.

Ahora, suponga que intercambiamos las columnas y_1 y y_2 de la asignación 1 para obtener la asignación 3 de la figura 9.18b. Tenemos las siguientes ecuaciones de excitación:

$$D_1 = y_2\bar{x} + \bar{y}_2 x$$
$$D_2 = y_2\bar{x} + y_1 x$$

Podemos deducir estas ecuaciones a partir de las ecuaciones de la asignación 1, intercambiando las variables y_1 y y_2 en las ecuaciones. El costo de la lógica necesaria para realizar estas ecuaciones continúa esencialmente sin cambio. Por tanto, la creación de una nueva asignación de estados al complementar o intercambiar las variables de estados no simplifica el circuito.

Para los circuitos secuenciales con sólo dos estados, en realidad sólo existe una elección de asignación de estados. Para tres o cuatro estados, las 24 asignaciones posibles se reducen a sólo tres; ver la tabla 9.2. Por tanto, es posible sintetizar el circuito secuencial usando las tres asignaciones y elegir la mejor. El número de asignaciones de estados posibles N_{UA} para un circuito secuencial con N_S estados está dado por

$$N_{UA} = \frac{(2^{N_{FF}} - 1)!}{(2^{N_{FF}} - N_S)!N_{FF}!}$$

La tabla 9.1 muestra varios casos para esta expresión. Si hay más de cuatro estados no resulta práctica la enumeración completa, por lo que necesitamos ciertas técnicas para elegir una buena asignación.

9.4.2 Criterios para la asignación de estados

Nuestro método para desarrollar un conjunto de criterios para la asignación de estados se basa en encontrar varias reglas que produzcan una asignación de estados con una menor complejidad de las ecuaciones de estado siguiente para la implantación de una tabla de estados reducida. El problema de la asignación de estados en este caso consiste en seleccionar una codificación de estados que obligue a la existencia de grandes agrupaciones de unos lógicos en la tabla de transición binaria o mapa K de transición. Cuanto más grandes sean los grupos, más podremos simplificar las ecuaciones de excitación y de salida y tendremos un circuito combinatorio menos complejo. Observe que la minimización lógica del estado siguiente sugiere el uso de *flip-flops* D, pues la entrada de excitación D_i es igual al estado siguiente Y_i para cada *flip-flop*.

TABLA 9.2 ASIGNACIONES DE ESTADO ÚNICAS

	Asignaciones		
Estados	1 y_1y_2	2 y_1y_2	3 y_1y_2
A	00	00	00
B	01	11	10
C	11	01	01
D	10	10	11

	x = 0	x = 1
A	$C/0$	$D/0$
B	$C/0$	$A/0$
C	$B/0$	$D/0$
D	$A/1$	$B/1$

Figura 9.19 Una máquina de cuatro estados.

Un ejemplo sencillo

Para tener una mejor idea de cómo determinar una buena asignación, examinemos un ejemplo de máquina de cuatro estados sintetizada mediante sus tres asignaciones de estados únicas, las cuales aparecen en la tabla 9.2. Consideremos la tabla de estados de la figura 9.19. Primero, realizaremos este circuito reducido con la asignación 1 y *flip-flops* D. Al sustituir la asignación 1 en la figura 9.19 obtenemos la tabla de transición de la figura 9.20a, que podemos reordenar como mapas K para z, D_2 y D_1, como vemos en las figuras 9.20b, c y d. Por tanto, las ecuaciones lógicas para esta asignación son:

Asignación 1.

$$D_2 = \bar{y}_1\bar{y}_2 + \bar{x}\bar{y}_2 + xy_1y_2$$
$$D_1 = \bar{x}\bar{y}_2 + \bar{x}y_1 + x\bar{y}_1y_2$$
$$z = \bar{y}_1y_2$$

Las figuras 9.21 y 9.22 ilustran los mismos procedimientos para las asignaciones 2 y 3. Las funciones booleanas de conmutación resultantes son las siguientes:

Asignación 2.

$$D_2 = x\bar{y}_1 + y_1\bar{y}_2$$
$$D_1 = \bar{x}\bar{y}_2 + \bar{x}y_1 + x\bar{y}_1y_2$$
$$z = \bar{y}_1y_2$$

y_2y_1 \ x	0	1
00	**11/0**	**10/0**
01	**11/0**	**00/0**
11	**01/0**	**10/0**
10	**00/1**	**01/1**

$Y_2\,Y_1/z$

(a)

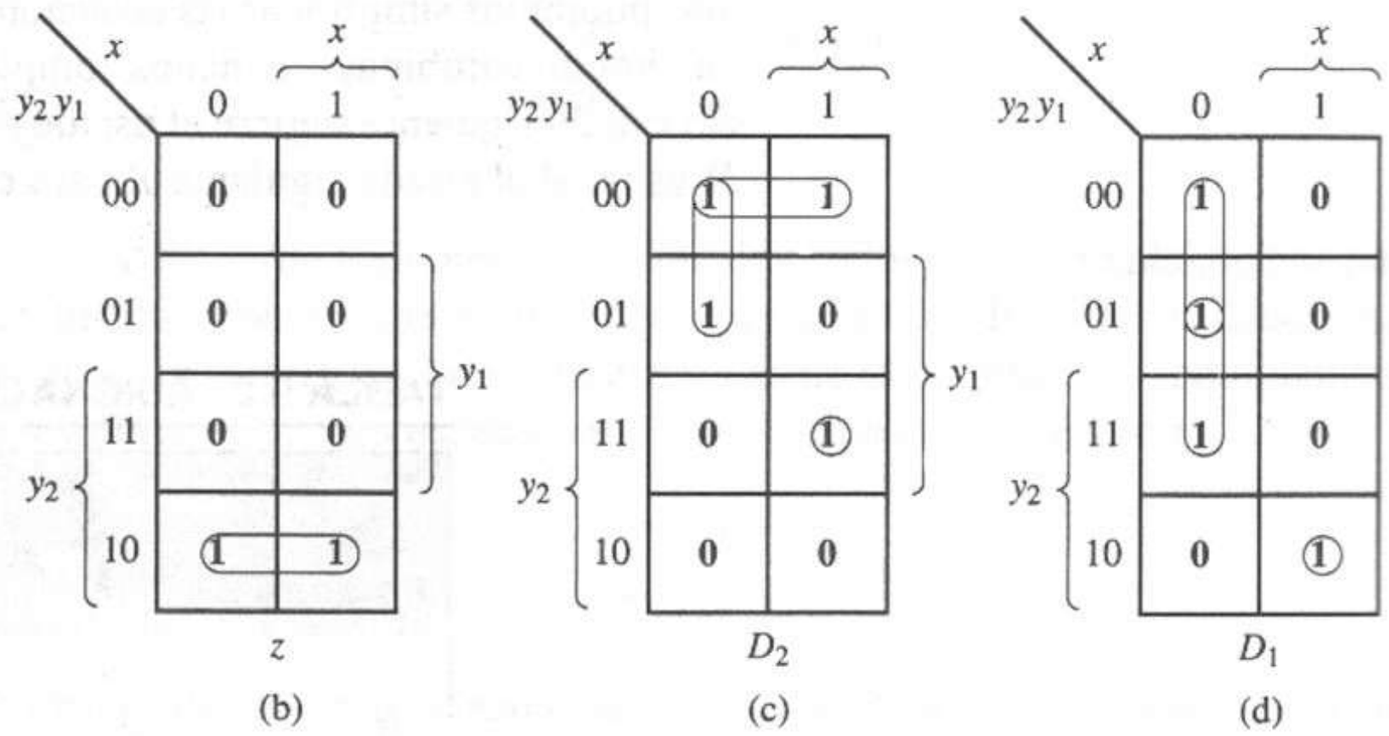


Figura 9.20 Realización con *flip-flop* D para la asignación 1. **(a)** Tabla de transición. **(b)** Mapa K para z. **(c)** Mapa K para D_2. **(d)** Mapa K para D_1.

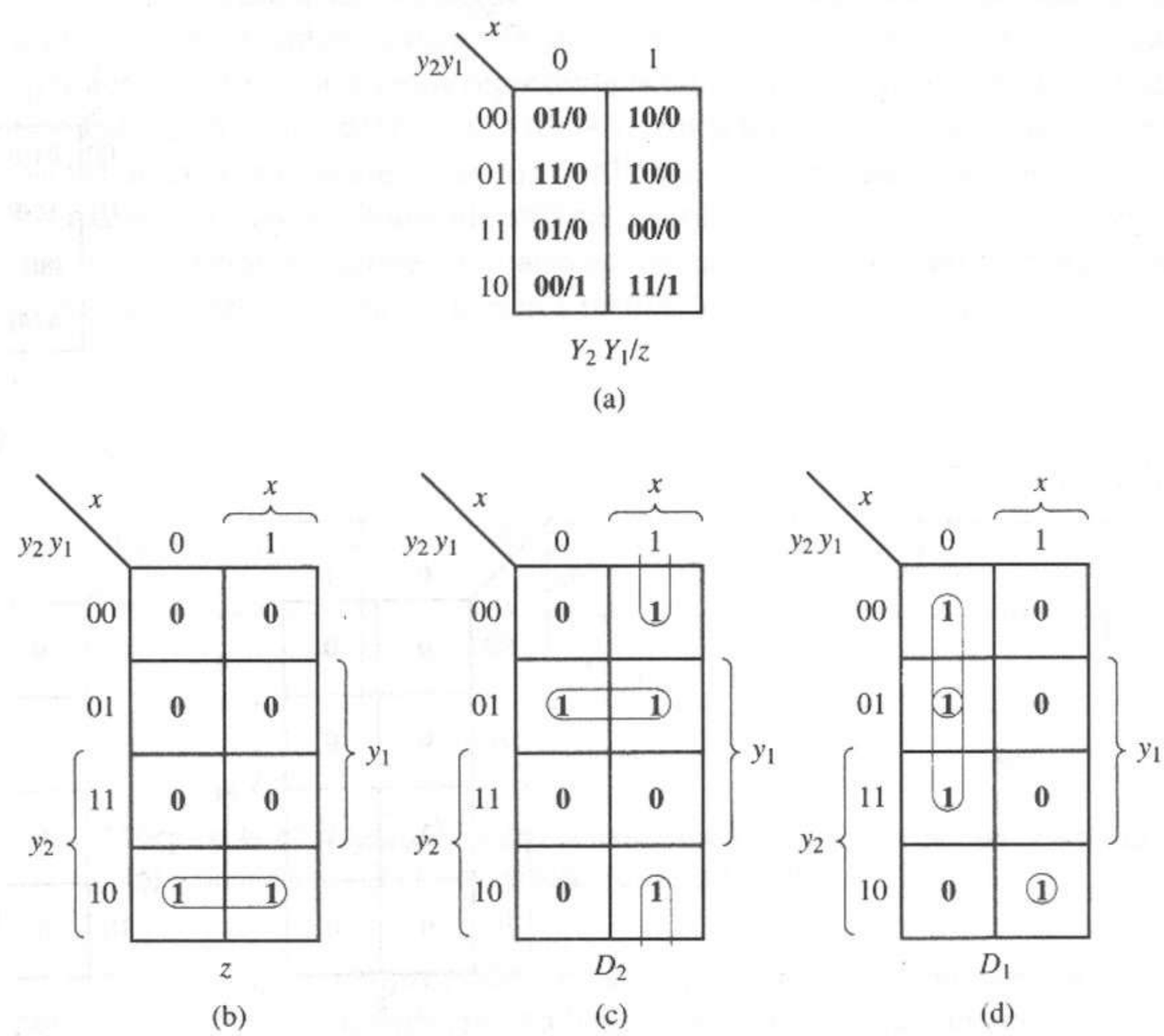


Figura 9.21 Realización con *flip-flop* D para la asignación 2. **(a)** Tabla de transición. **(b)** Mapa K para z. **(c)** Mapa K para D_2. **(d)** Mapa K para D_1.

Asignación 3.

$$D_2 = y_1\bar{y}_2 + x\bar{y}_2 + xy_1$$
$$D_1 = x\bar{y}_2 + \bar{x}\bar{y}_1$$
$$z = y_1y_2$$

Examinemos ahora los resultados. Si especificamos una implantación lógica de dos niveles de suma de productos y contamos el número de entradas a las compuertas, la asignación 1 requiere 20; la asignación 2, 18; y la asignación 3, 15. La asignación 3 da el mejor resultado pero, ¿por qué? La asignación 3 proporciona una mejor agrupación de los unos y ceros en los mapas K para D_1, D_2 y z.

Reglas para la asignación de estados

Existen dos formas de reordenar los unos de un mapa K: en forma vertical, haciendo que los unos se combinen en una columna dada, y en forma horizontal, haciendo que los unos se combinen en una fila dada. Si suponemos que cada fila representa las entradas de los estado siguiente para un estado actual dado y que las columnas representan diferentes condiciones de entrada, entonces podemos obtener agrupaciones óptimas de unos si adoptamos las siguientes reglas generales para la asignación de estados.

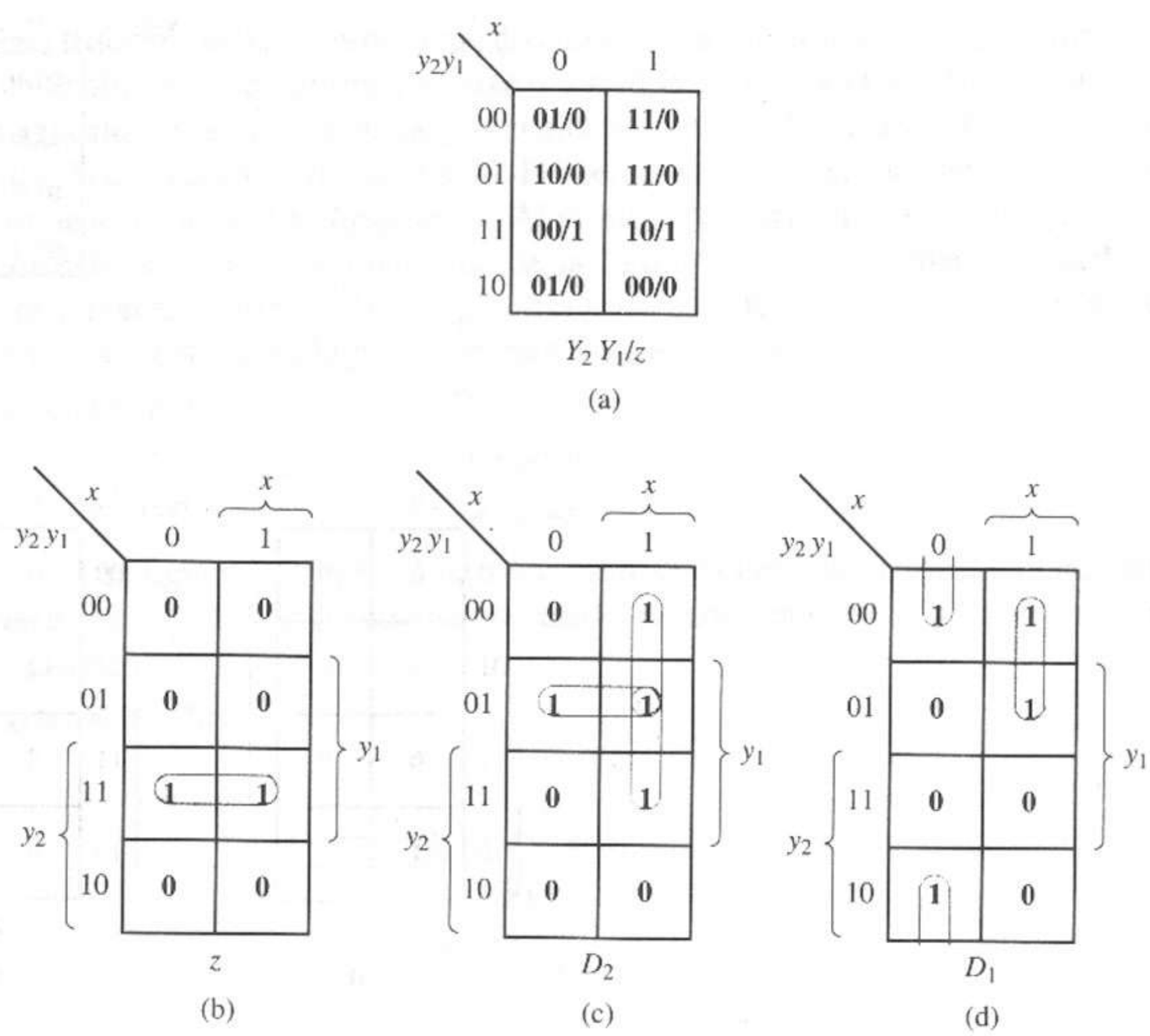


Figura 9.22 Realización con *flip-flop* D para la asignación 3. **(a)** Tabla de transición. **(b)** Mapa K para z. **(c)** Mapa K para D_2. **(d)** Mapa K para D_1.

Regla 1. Los estados con los mismos estados siguientes para una entrada dada deben recibir asignaciones lógicamente adyacentes.

Regla 2. Los estados que son estados siguientes de un único estado actual, bajo entradas lógicamente adyacentes, deben recibir asignaciones lógicamente adyacentes.

La regla 1 hace que los unos se agrupen en una columna dada, mientras que la regla 2 hace que los unos se agrupen en una fila dada. Podemos aplicar ambas reglas al inspeccionar la tabla de estados reducida. Ilustraremos esto mediante el siguiente ejemplo.

EJEMPLO 9.17

Aplicar las reglas anteriores al ejemplo de la figura 9.19.

Regla 1: Los estados A y B deben ser adyacentes, ya que ambos van al estado C bajo la entrada 0; denotamos esto con A ady B. También indicamos A ady C.

Regla 2: Produce A ady B, A ady C, B ady D y C ady D.

Podemos comparar las adyacencias de los estados para las tres asignaciones graficando los estados en *mapas K de asignación de estados*, como muestra la figura 9.23, donde cada celda representa la combinación de variables de estado asignadas a un estado del circuito. Identificamos los estados adyacentes en estos mapas de la misma manera que los implicantes o implicados adyacentes sobre los mapas K estándar. Podemos utilizar los mapas K de la figura 9.23 para ver que la asignación 3 cumple las cuatro adyacencias indicadas en las reglas 1 y 2 y, por tanto, proporciona los mejores resultados para este ejemplo.

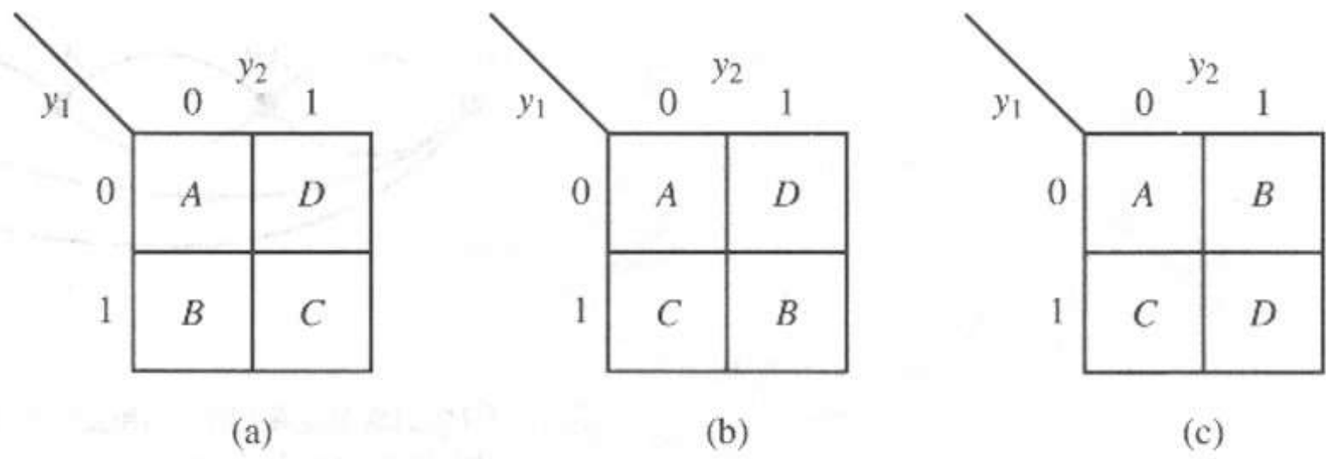


Figura 9.23 Adyacencias de estados para asignaciones de cuatro estados. **(a)** Asignación 1. **(b)** Asignación 2. **(c)** Asignación 3.

En general, no siempre podemos satisfacer todas las adyacencias sugeridas por las reglas 1 y 2. Sin embargo, podemos tratar de satisfacer el mayor número posible. En caso de conflictos, resolvemos éstos a favor de la regla 1 para obtener una mejor agrupación de unos dentro de las columnas.

Gráfica de implicación

Otra herramienta que ayuda al diseñador lógico en la elección de buenas asignaciones de estados es la gráfica de implicación. La *gráfica de implicación* para un circuito secuencial es una gráfica de flujo cuyos nodos representan pares de estados. Los nodos se conectan mediante arcos, cada uno de los cuales representa transiciones de estado entre dos pares de estados para una entrada dada, según lo especificado por la tabla de estados del circuito secuencial.

EJEMPLO 9.18

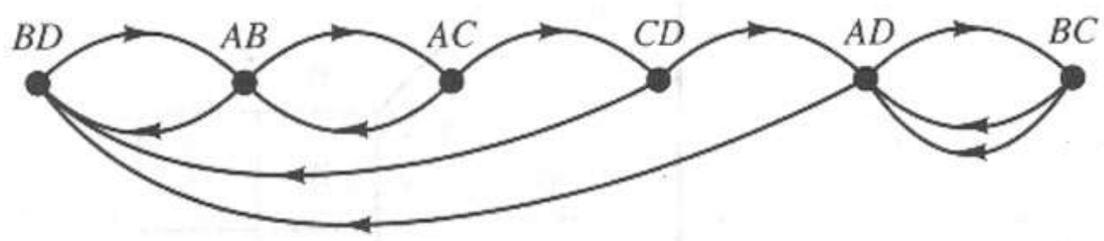
Deducir una gráfica de implicación para el circuito secuencial de la figura 9.24a.

La figura 9.24b muestra la gráfica de implicación para este circuito.

Una gráfica de implicación se asemeja mucho a la tabla de implicación presentada con anterioridad y contiene casi la misma información; sin embargo, aquí se despliega la información de manera gráfica. Construimos la gráfica de implicación eligiendo primero un par de estados, digamos *BD* en el ejemplo, y determinando su par de estados siguientes implicado bajo cada entrada; los pares implicados se introducen como nuevos nodos en la gráfica. No introducimos un par de estados en la gráfica si los dos estados tienen el mismo estado siguiente. Por ejemplo, los estados *B* y *D* van al estado *D* bajo una entrada igual a 0 y a los estados *A* y *B* bajo una entrada igual a 1. En consecuencia, indicamos una

	x = 0	x = 1
A	$B/0$	$C/0$
B	$D/0$	$A/1$
C	$A/1$	$D/0$
D	$D/1$	$B/1$

(a)



(b)

Figura 9.24 Una máquina de cuatro estados. **(a)** Tabla de estados. **(b)** Gráfica de implicación.

transición de par de estados de BD a AB para la entrada $x = 1$, pero no indicamos una entrada para $x = 0$. Ahora repetimos el procedimiento para cada nuevo par implicado hasta que no podamos generar más pares implicados.

Una gráfica de implicación es *completa* si contiene todos los pares de estados posibles para un circuito secuencial dado. Por lo general, tenemos una gráfica de implicación parcialmente completa. Una *subgráfica* es parte de una gráfica completa.

Un tipo particular de subgráfica, la gráfica *cerrada,* es importante para la asignación de estados. Una subgráfica es *cerrada* si todos los arcos de salida para cada nodo dentro de la subgráfica terminan en nodos totalmente contenidos dentro de la subgráfica y si cada estado de un circuito secuencial está representado por al menos un nodo dentro de la subgráfica.

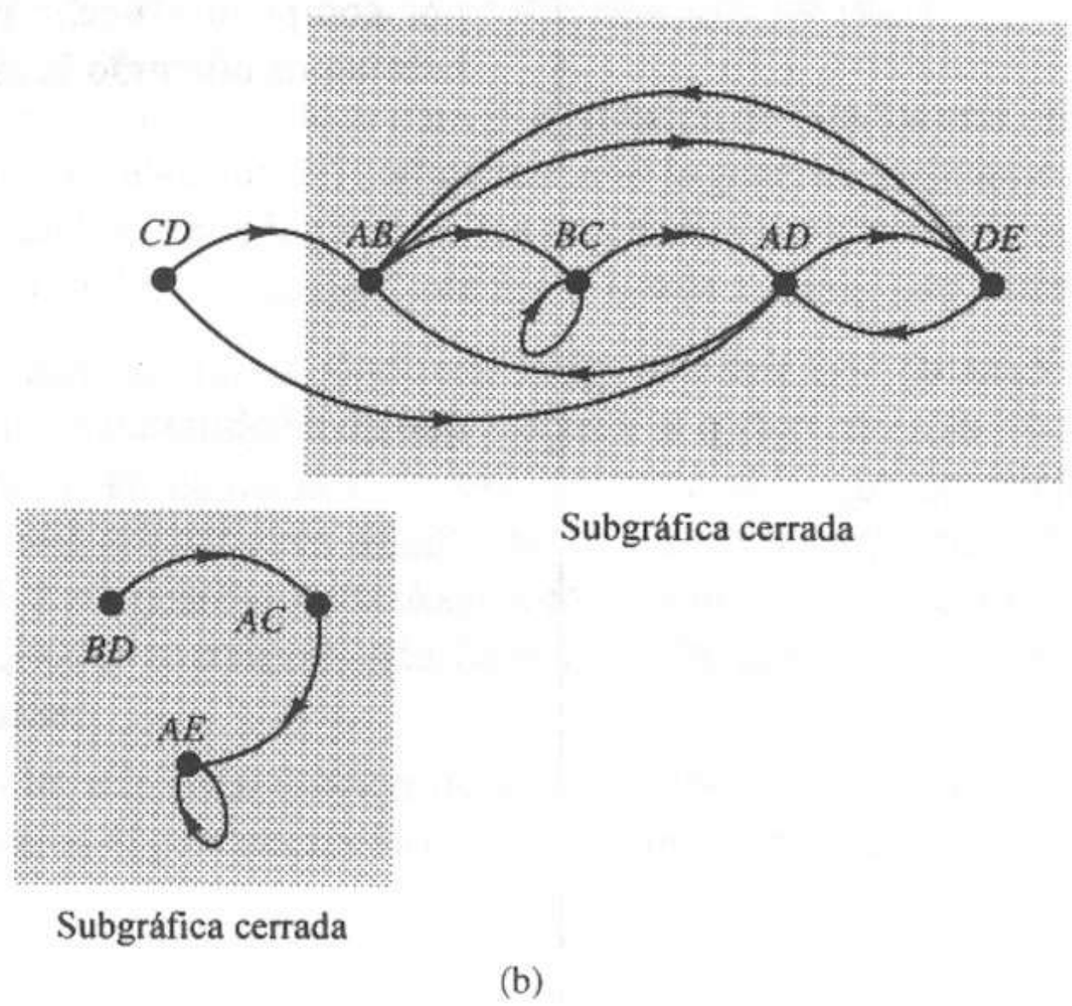
EJEMPLO 9.19

La figura 9.25b muestra dos subgráficas cerradas para el circuito secuencial de la figura 9.25a. Observe que los arcos pueden *entrar* en una subgráfica cerrada desde el exterior, pero ninguno puede originarse dentro de la subgráfica cerrada y salir.

Podemos utilizar la gráfica de implicación para elegir las asignaciones de estado de la manera siguiente. Después de aplicar las reglas 1 y 2 a una tabla de estados reducida, tenemos varias adyacencias sugeridas. Podemos elegir algunas de éstas para implantarlas en una asignación de estados. Una vez que hemos hecho a dos estados adyacentes lógicamente, es posible reordenar la tabla de estados para que los dos estados estén adyacentes físicamente. Si consideramos estos dos estados adyacentes como estados actuales del circuito secuencial, es recomendable que sus pares de estados siguientes se hagan adyacentes, para

	x = 0	x = 1
A	$B/0$	$E/0$
B	$C/1$	$D/1$
C	$B/0$	$A/0$
D	$A/0$	$D/0$
E	$B/1$	$A/1$

(a)



(b)

Figura 9.25 Subgráficas cerradas. **(a)** Tabla de estados. **(b)** Gráfica de implicación.

obtener grupos de unos mayores en la tabla de transición. El siguiente ejemplo ilustra este punto.

EJEMPLO 9.20

Utilizar la gráfica de implicación de la figura 9.24b para deducir una asignación de estados óptima para el circuito secuencial de la figura 9.24a.

Al aplicar las reglas 1 y 2 a la tabla de estados obtenemos

Regla 1. *B* ady *D*.

Regla 2. *B* ady *C*, *D* ady *A*, *A* ady *D*, *D* ady *B*.

La adyacencia *B* ady *D* es la más importante. Si hacemos al estado *B* adyacente al estado *D*, es recomendable hacer *A* ady *B* como lo indica la gráfica de implicación. La adyacencia no fue sugerida por la regla 1 o 2. De hecho, la gráfica de implicación también sugiere *A* ady *C* ady *D*. Estas adyacencias corresponden a la asignación 3 de la figura 9.23c. La figura 9.26a muestra la

tabla de transición para esta asignación. Las funciones de conmutación de excitación y de salida resultantes son

$$D_2 = \bar{x}y_2 + \bar{x}\bar{y}_1 + xy_1$$
$$D_1 = \bar{x}y_2 + x\bar{y}_2$$
$$z = xy_2 + \bar{x}y_1$$

Estas funciones de conmutación son óptimas para este circuito secuencial. La clave del éxito para lograr una asignación de estados adecuada fue la obtención de A ady B mediante la gráfica de implicación. Si hubiéramos ignorado la gráfica por completo y hecho nuestra asignación sólo con base en las reglas 1 y 2, habríamos obtenido la asignación 2, y la realización hubiera sido

$$D_2 = xy_1\bar{y}_2 + \bar{x}\bar{y}_1 + \bar{x}y_2 + \bar{y}_1y_2$$
$$D_1 = \bar{y}_1\bar{y}_2 + x\bar{y}_1$$
$$z = xy_2 + \bar{y}_1y_2 + \bar{x}y_1\bar{y}_2$$

que no es óptima.

y_2y_1 \ x	0	1
$A \rightarrow$ 00	10/0	01/0
$C \rightarrow$ 01	00/1	11/0
$D \rightarrow$ 11	11/1	10/1
$B \rightarrow$ 10	11/0	00/1

Y_2Y_1/z

(a)

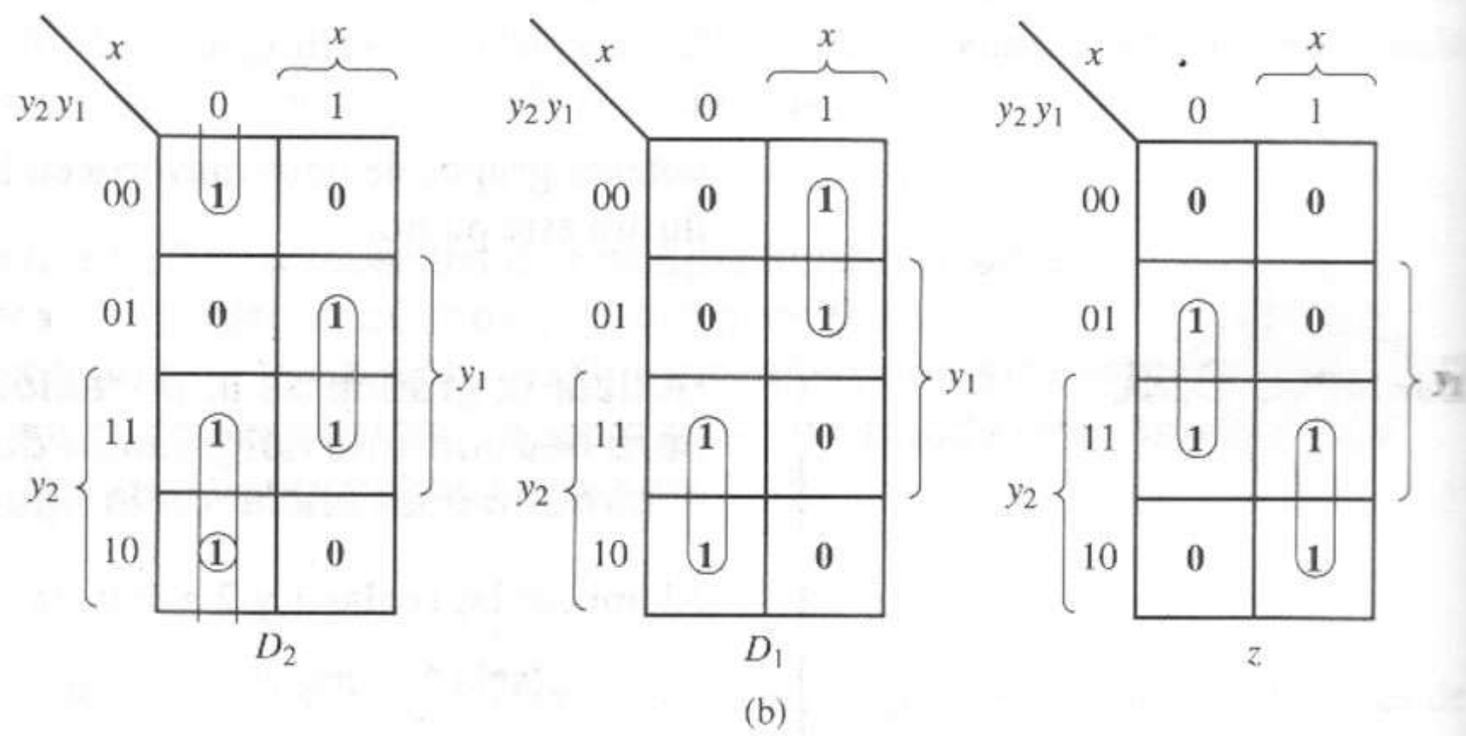


Figura 9.26 Ejemplo de asignación de estados. **(a)** Tabla de transición. **(b)** Mapas K.

El ejemplo 9.20 ilustra la forma de establecer una *cadena* de pares de adyacencias. Establecemos la cadena al asignar los pares de estados adyacentes de acuerdo con los arcos de transición de la gráfica de implicación. Las cadenas

ideales son las establecidas mediante una subgráfica cerrada. Si no podemos establecer una subgráfica cerrada, hay que intentar incluir una subgráfica contigua que contenga un número mayor de arcos de transición.

Ahora incorporaremos la gráfica de implicación a nuestros criterios de asignación de estados mediante la siguiente regla.

Regla 3. Utilice las adyacencias sugeridas por las reglas 1 y 2 para construir una gráfica de implicación parcialmente completa para la tabla de estados reducida. Después, intente establecer una cadena de pares de adyacencia en una subgráfica cerrada o contigua.

Al aplicar las reglas 1, 2 y 3, cabe destacar que las reglas 1 y 3 son más importantes que la regla 2.

El último criterio por analizar para la asignación de estados está relacionado con el mecanismo de elección del código real. Por lo general, intentamos minimizar el número total de unos lógicos en los mapas K y maximizar el número de condiciones prescindibles. La siguiente regla ayuda a lograr esta meta:

Regla 4. Busque en la porción de estado siguiente de la tabla de estados reducida el estado "más transferido hacia". Asígnele el código con sólo ceros lógicos, colocándolo en un mapa K de asignación en el bloque del mintérmino 0. Comience a asignar otros estados según las adyacencias sugeridas por las reglas 1, 2 y 3, guardando los bloques de mintérminos con el mayor número de unos lógicos en su código binario. Satisfaga tantas adyacencias sugeridas como sea posible.

Esto concluye la presentación de los criterios de asignación de estados. Ahora ilustraremos el procedimiento mediante un ejemplo completo.

EJEMPLO 9.21

Determinar una realización con *flip–flops* D para el circuito secuencial de la figura 9.27a.

Regla 1. A ady C, B ady D, A ady D.

Regla 2. A ady D, B ady E, A ady B, C ady D.

Regla 3. De las reglas 1 y 2 trazamos la gráfica de implicación de la figura 9.27b e identificamos dos subgráficas cerradas, una de las cuales contiene A ady D, A ady E, B ady C, D ady E y la segunda, sólo B ady D. Observe que hemos agregado el par CE para que la gráfica sea completa, es decir, para mostrar todos los pares de estados. En ejemplos de mayor tamaño la generación de una gráfica completa puede hacer sea demasiado compleja la evaluación a mano de la gráfica. En tales casos, limitamos la gráfica a los nodos generados mediante las reglas 1 y 2.

Regla 4. La figura 9.27c muestra el mapa K de asignación de estados.

Con esta asignación de estados, podemos generar las siguientes funciones de conmutación, como se muestra en la figura 9.28:

$$D_3 = x\bar{y}_3$$
$$D_2 = x\bar{y}_3\bar{y}_1 + xy_2 + \bar{x}y_1$$
$$D_1 = \bar{x}\bar{y}_2\bar{y}_1$$
$$z = xy_2 + y_3y_2$$

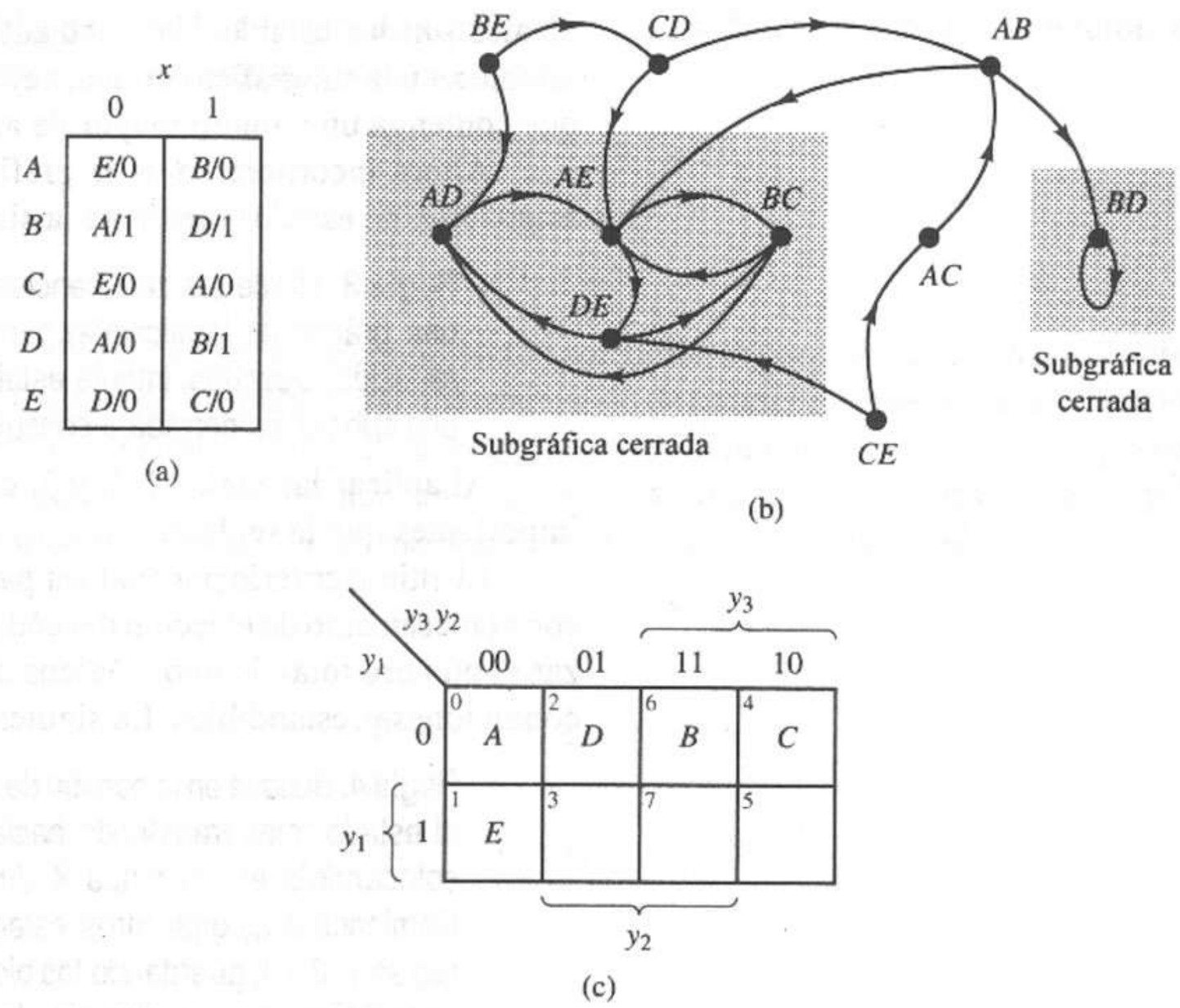


Figura 9.27 Ejemplo de asignación de estados. **(a)** Tabla de estados. **(b)** Subgráficas cerradas. **(c)** Asignación de estados.

Lógica de salida

Debemos mencionar las ecuaciones de salida. En nuestro procedimiento de asignación de estados hemos ignorado por completo cualquier consideración de minimización de la lógica de salida. Si el número de ecuaciones booleanas de estado siguiente es mucho mayor que el número de ecuaciones de salida, aplique las reglas según lo indicado por los estados siguientes y acepte los resultados de las ecuaciones de salida. Sin embargo, si las ecuaciones de salida son una porción significativa de la lógica de implantación, las reglas 1 y 2 son igualmente aplicables para minimizar la lógica de salida. Sin embargo, en este texto nos restringiremos al análisis de los estados siguientes.

9.4.3 Partición

Recuerde que una *partición* P_i del conjunto de estados de una máquina secuencial es una colección de subconjuntos ajenos B_{ij} que contienen a todos los estados de la máquina. El proceso de formación de una asignación de estados para la máquina secuencial equivale a crear una serie de particiones con dos bloques (llamadas *particiones de dos bloques*) del conjunto de estados, con una partición para cada *flip-flop* de memoria.

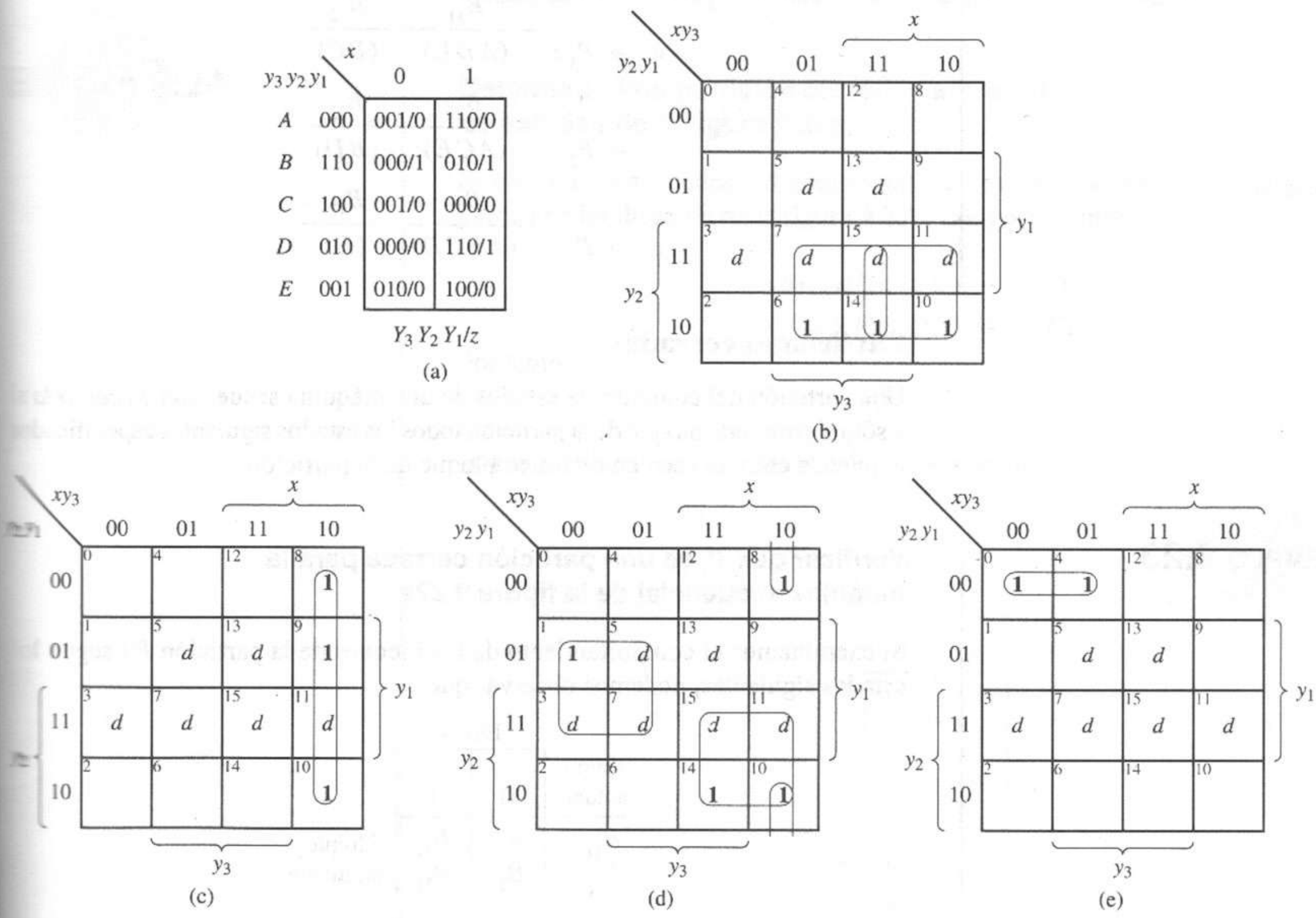


Figura 9.28 Realización con *flip-flop* D. **(a)** Tabla de transición. **(b)** Mapa K para z. **(c)** Mapa K para D_3. **(d)** Mapa K para D_2. **(e)** Mapa K para D_1.

EJEMPLO 9.22

Determinar las particiones de dos bloques creadas en la máquina secuencial de la figura 9.27a por la asignación de la figura 9.27c.

El código asignado es

	y_3	y_2	y_1
A:	0	0	0
B:	1	1	0
C:	1	0	0
D:	0	1	0
E:	0	0	1

Cada columna de bits de la asignación separa los estados en dos bloques o subconjuntos: aquellos asociados con 0 lógico y los asociados con 1 lógico.

	B_{31}	B_{32}
$y_3 \rightarrow P_3$:	(ADE)	(BC)

	B_{21}	B_{22}
$y_2 \rightarrow P_2$:	(ACE)	(BD)

	B_{11}	B_{12}
$y_1 \rightarrow P_1$:	$(ABCD)$	(E)

Particiones cerradas

Una partición del conjunto de estados de una máquina secuencial es *cerrada* si y sólo si para cada bloque de la partición todos los estados siguientes especificados bajo cada entrada caen en un único bloque de la partición.

EJEMPLO 9.23

Verificar que P_3 es una partición cerrada para la máquina secuencial de la figura 9.27a.

Si examinamos el comportamiento de los bloques de la partición $P3$ según los estados siguientes, podemos observar que

Bloque actual	Entrada 0	Entrada 1	
B_{31}	B_{31}	B_{32}	Bloque siguiente
B_{32}	B_{31}	B_{31}	

Los estados siguientes de los bloques de P_3 también están contenidos en los bloques de P_3. Por tanto, la partición P_3 es cerrada.

La presencia de particiones cerradas en los estados de una máquina secuencial puede reducir en gran medida la dependencia de las ecuaciones de los estados siguientes Y_i $(i = 1,...,r)$ con respecto de las variables de estados actuales y_i $(i = 1,...,r)$. Si una partición cerrada tiene dos bloques, podemos utilizar una de las variables de estado para codificar los bloques; es decir, podemos asignar $y_i = 0$ para todos los estados de uno de los bloques y $y_i = 1$ para todos los estados del otro bloque; esta variable es el *bit de bloque*. La presencia de la partición cerrada hace que la variable de estado siguiente Y_j sólo dependa del bit de bloque yj y de las entradas del circuito; es decir,

$$Y_j = h_j(x_1, x_2, \ldots, x_n, y_j)$$

Ejemplificamos lo anterior deduciendo la ecuación de estado siguiente para Y_3 en la asignación de estados de la figura 9.27c:

$$Y_3 = x\bar{y}_3$$

Podemos generar particiones cerradas mediante una gráfica de implicación. Comenzamos suponiendo que dos estados S_i y S_j están en el mismo bloque de la partición. Por la definición de una partición cerrada, los pares de estados

siguientes implicados de S_iS_j también deben estar juntos en algún bloque de la partición cerrada. A continuación, combinamos los pares implicados en bloques ajenos, utilizando las propiedades de las relaciones de equivalencia.

EJEMPLO 9.24

Determinar una partición cerrada para la tabla de estados de la figura 9.29a.

Si elegimos los estados AB como punto de partida, generamos la subgráfica parcial de implicación de la figura 9.29b. Los pares implicados son

$$(AB), (BC), (AC) \rightarrow (ABC)$$
$$(DE), (EF), (DF) \rightarrow (DEF)$$

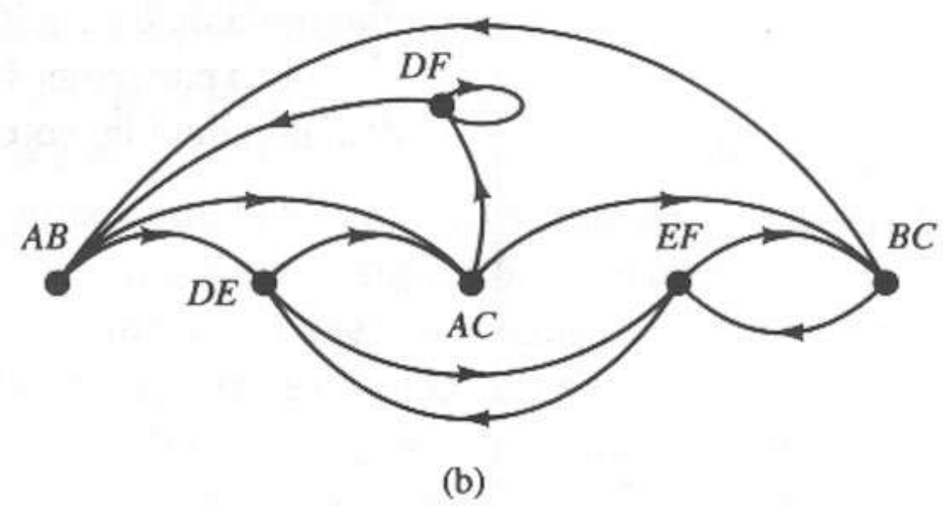
Por tanto,

$$P_1 = (ABC)(DEF)$$

es una partición cerrada para la máquina secuencial.

	x = 0	x = 1
A	$D/0$	$C/0$
B	$E/0$	$A/1$
C	$F/1$	$B/0$
D	$A/1$	$F/1$
E	$C/0$	$E/0$
F	$B/0$	$D/1$

(a)



(b)

Figura 9.29 Particiones cerradas. **(a)** Tabla de estados. **(b)** Gráfica de implicación parcial.

La partición cerrada describe una propiedad existente en ciertas máquinas. Si la partición cerrada tiene dos bloques y la asignación de estados codifica estos bloques con un bit de bloque, obtenemos las ecuaciones lógicas reducidas de los estados siguientes. La regla 3 del procedimiento de asignación de estados ayuda al diseñador lógico a determinar las asignaciones de estado correspondientes a las particiones cerradas sin que haya que generar éstas.

Pares de particiones

Ciertas máquinas secuenciales exhiben un segundo tipo de dependencia reducida, llamada *dependencia cruzada*. Estas máquinas se basan en pares de particiones que no necesariamente son cerradas. Las transiciones de estado de las máquinas secuenciales con dependencia cerrada quedan descritas por transiciones de bloque entre pares de particiones de bloque, como muestra el siguiente ejemplo.

EJEMPLO 9.25

La máquina secuencial de la figura 9.17 exhibe una dependencia cruzada bajo la asignación minimal

	y_3	y_2	y_1
A:	0	0	0
B:	0	0	1
C:	0	1	1
D:	0	1	0
E:	1	0	1
F:	1	1	0
G:	1	1	1

Las particiones de dos bloques para la asignación son

	B_{11}	B_{12}
P_1:	$(ABCD)$	(EFG)

	B_{21}	B_{22}
P_2:	(ABE)	$(CDFG)$

	B_{31}	B_{32}
P_3:	(ADF)	$(BCEG)$

Un análisis minucioso del comportamiento de los bloques revela la tabla y diagrama de transición de bloques de la figura 9.30.

Las particiones P_2 y P_3 no son cerradas, pero como par exhiben una dependencia reducida, como lo muestra la siguiente implantación con un *flip–flop* D:

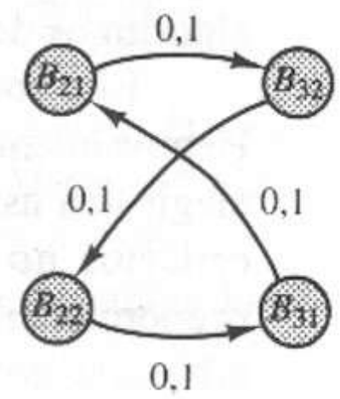
$$D_1 = x\bar{y}_1 y_3 + \bar{x} y_1 y_3 + x\bar{y}_1 \bar{y}_2$$
$$\left.\begin{aligned} D_2 &= y_3 \\ D_3 &= \bar{y}_2 \end{aligned}\right\} \text{dependencia cruzada}$$
$$z = x y_1 \bar{y}_3 + \bar{x}\bar{y}_1 y_2 \bar{y}_3$$

La generación de pares de particiones está más allá de los objetivos de este libro; el lector interesado puede consultar las referencias al final del capítulo. La regla 3 del procedimiento de asignación de estados ya presentado ayuda al diseñador lógico a determinar asignaciones con dependencia cruzada, si éstas existen. Aunque los criterios para la asignación de estados no son óptimos, pueden producir buenas asignaciones de estado, incluso para máquinas con dependencia reducida.

El lector debe observar que la dependencia cruzada, que se basa en un par de particiones, es una generalización de la dependencia reducida, la cual se basa

Bloque actual	Entrada 0	Entrada 1
P_2: B_{21}	B_{32}	B_{32}
B_{22}	B_{31}	B_{31}
P_3: B_{31}	B_{21}	B_{21}
B_{32}	B_{22}	B_{22}

(a)



(b)

Figura 9.30 Dependencia cruzada. **(a)** Tabla de transición de bloques. **(b)** Diagrama de transición de bloques.

en una única partición cerrada. La dependencia reducida se puede considerar como un caso particular de la dependencia cruzada, en donde el par de particiones se forma utilizando una partición cerrada dos veces.

9.4.4 Asignaciones óptimas de estados

Muchos autores han propuesto procedimientos para obtener asignaciones de estados "buenas" o "casi óptimas", y varios afirman que sus procedimientos proporcionan asignaciones "óptimas" según ciertos criterios de optimalidad establecidos. A continuación enumeramos varios criterios de optimalidad comunes:

1. Circuito minimal con respecto de las compuertas o las entradas a compuertas
2. Circuito de costo mínimo
3. Circuito con dependencia reducida
4. Los criterios 1 a 3 para un tipo específico de *flip-flop*

Dolotta y McCluskey [9], Weiner y Smith [10] y Torng [11] han desarrollado varios de los mejores métodos para obtener asignaciones casi óptimas para los *flip-flops* D. Estos métodos buscan la dependencia reducida en la tabla de estados dada para una máquina secuencial.

Hasta este momento sólo hemos analizado el caso de los *flip-flops* D. ¿Qué ocurre si queremos utilizar otro tipo? En general, una *buena* asignación de estados para los *flip-flops* D será una *buena* asignación para otros tipos de *flip-flop*. Sin embargo, Curtis [12] ha mostrado que una asignación óptima o casi óptima para un tipo de *flip-flop* puede estar lejos de ser la óptima para otro. Curtis ha extendido los métodos de Dolotta y McCluskey y Weiner y Smith a la obtención de asignaciones de estado casi óptimas para tipos específicos de *flip-flops* o combinaciones de *flip-flops* de diversos tipos dentro de una realización de máquina secuencial.

Para obtener asignaciones óptimas, la complejidad computacional de los algoritmos propuestos para la asignación de estados requiere el uso de una computadora digital de uso general. Story *et al.* [13] y Haring [14] han desarrollado tales algoritmos óptimos. Story *et al.* determinan una asignación óptima de estados para los *flip-flops* JK con reloj.

Para la mayor parte de las aplicaciones, no se requiere una solución óptima o casi óptima al problema de asignación de estados; una buena asignación es adecuada en la mayor parte de los casos. Sin embargo, si la asignación óptima

es esencial el diseñador lógico puede consultar la bibliografía relativa a los algoritmos de asignación de estados que garantizan la optimalidad.

Esto concluye nuestra presentación del problema de asignación de estados. Hemos intentado formular ciertos criterios generales que ayuden al diseñador a elegir una asignación de estados para una máquina secuencial dada. Nuestros criterios no garantizan una realización lógica mínima; sin embargo, sí proporcionan por lo general un mejor resultado que una elección totalmente arbitraria, como lo habíamos hecho antes en el texto.

9.5 Resumen

En este capítulo presentamos los procedimientos de síntesis para realizar circuitos secuenciales síncronos con especificación completa o incompleta. Demostramos la reducción de estados mediante inspección, partición y la tabla de implicación. Contrastamos los conceptos de equivalencia de estados para los circuitos con especificación completa y compatibilidad de estados para los circuitos con especificación incompleta. Prestamos atención a las tablas de entrada de los *flip-flops* y a la generación de mapas de excitación, al igual que al problema de asignación de estados. Realizamos muchos ejemplos de diseño para ilustrar las técnicas de síntesis. El lector debe tener una buena idea del problema de síntesis de un circuito secuencial síncrono, pero si desea mayor información acerca del problema de síntesis puede consultar las referencias enumeradas a continuación. Además, se describen métodos asistidos por computadora en Downs y Schulz [15], y están implantados en varios sistemas CAD, como *Logic Aid* de Roth [16].

REFERENCIAS BIBLIOGRÁFICAS

1. E. J. McCluskey, *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965.
2. Zvi Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill Book Co., 1970.
3. Taylor L. Booth, *Digital Networks and Computer Systems*. New York: Wiley, 1971.
4. J. Hartmanis, "On the State Assignment Problem for Sequential Machines I," *IRE Trans. Electronic Computers*, June 1961, pp. 157–165.
5. R. E. Stearns and J. Hartmanis, "On the State Assignment Problem for Sequential Machines II," *IRE Trans. Electronic Computers*, December 1961, pp. 593–603.
6. M. C. Paull and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," *IRE Trans. Electronic Computers*, EC-8, No. 3, September 1959, pp. 356–357.
7. E. J. McCluskey and S. H. Unger, "A Note on the Number of Internal Variable Assignments for Sequential Switching Circuits," *IRE Trans. Electronic Computers*, EC-8, No. 4, December 1959, pp. 439–440.
8. R. M. Karp, "Some Techniques of State Assignment for Synchronous Sequential Machines," *IEEE Trans. Electronic Computers*, EC-13, No. 5, October 1964, pp. 507–518.

9. T. A. DOLOTTA AND E. J. MCCLUSKEY, "The Coding of Internal States of Sequential Circuits," *IEEE Trans. Electronic Computers*, EC-13, No. 5, October 1964, pp. 549–562.

10. P. WEINER AND E. J. SMITH, "Optimization of Reduced Dependences for Synchronous Sequential Machines," *IEEE Trans. Electronic Computers*, EC-16, No. 6, December 1967, pp. 835–847.

11. H. C. TORNG, "An Algorithm for Finding Secondary Assignments of Synchronous Sequential Circuits," *IEEE Trans. Computers*, C-17, No. 5, May 1968, pp. 461–469.

12. H. A. CURTIS, "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-flop Memory: Part I," *IEEE Trans. Computers*, C-18, No. 12, December 1969, pp. 1121–1127.

13. J. R. STORY, H. J. HARRISON, AND E. A. REINHARD, "Optimum State Assignment for Synchronous Sequential Circuits," *IEEE Trans. Computers*, C-21, No. 12, December 1972, pp. 1365–1373.

14. D. R. HARING, *Sequential Circuit Synthesis: State Assignment Aspects*, M.I.T. Research Monograph No. 31. Cambridge, MA: The M.I.T. Press, 1966.

15. T. DOWNS AND M. F. SCHULZ, *Logic Design with Pascal— Computer-aided Design Techniques*. New York: Van Nostrand Reinhold. 1988.

16. C. H. ROTH JR., *User's Guide and Reference Manual for LogicAid*. St. Paul, MN: West Publishing Co., 1992.

PROBLEMAS

9.1 Determine una tabla de estados minimizada para el siguiente circuito secuencial síncrono mediante

(a) Inspección.

(b) Partición.

(c) Tabla de implicación.

	I	J
A	$B/0$	$A/1$
B	$C/0$	$A/0$
C	$C/0$	$B/0$
D	$E/0$	$D/1$
E	$C/0$	$D/0$

9.2 Reduzca las siguientes tablas de estado mediante inspección.

(a)

	I	J
A	$B/1$	$C/0$
B	$A/1$	$C/0$
C	$D/1$	$A/0$
D	$C/1$	$A/1$

(b)

	I	J
A	$A/0$	$E/1$
B	$E/1$	$C/0$
C	$A/1$	$D/1$
D	$F/0$	$G/1$
E	$B/1$	$C/0$
F	$F/0$	$E/1$
G	$A/1$	$D/1$

(c)

	I	J	K
A	$A/0$	$B/1$	$E/1$
B	$B/0$	$A/1$	$F/1$
C	$A/1$	$D/0$	$E/0$
D	$F/0$	$C/1$	$A/0$
E	$A/0$	$D/1$	$E/1$
F	$B/0$	$D/1$	$F/1$

9.3 Reduzca las tablas de estados del problema 9.2 por partición.

9.4 Reduzca las tablas de estados del problema 9.2 utilizando tablas de implicación.

9.5 Elabore una tabla de estados reducida para el siguiente circuito secuencial síncrono:

	I	J
A	$B/0$	$C/0$
B	$D/0$	$E/0$
C	$F/0$	$G/0$
D	$A/1$	$B/1$
E	$C/0$	$D/0$
F	$F/0$	$G/0$
G	$B/0$	$F/0$

9.6 Utilice una tabla de implicación para reducir el siguiente circuito secuencial a un número mínimo de estados:

	I	J
A	$A/0$	$C/0$
B	$D/1$	$A/0$
C	$F/0$	$F/0$
D	$E/1$	$B/0$
E	$G/1$	$G/0$
F	$C/0$	$C/0$
G	$B/1$	$H/0$
H	$H/0$	$C/0$

9.7 Reduzca el número de estados del siguiente circuito secuencial:

(a) Por partición. **(b)** Mediante una tabla de implicación.

	I	J	K
A	$D/1$	$C/0$	$E/1$
B	$D/0$	$E/0$	$C/1$
C	$A/0$	$E/0$	$B/1$
D	$A/1$	$B/0$	$E/1$
E	$A/1$	$C/0$	$B/1$

9.8 Obtenga una realización mediante *flip–flops* D con reloj para el siguiente circuito secuencial usando cada una de las tres asignaciones únicas para los circuitos con cuatro estados:

	x	
	0	1
A	$B/0$	$D/0$
B	$C/0$	$A/0$
C	$D/0$	$A/0$
D	$B/1$	$C/1$

9.9 Para la siguiente tabla de estados, obtenga implantaciones del circuito con cada una de las tres asignaciones de estado únicas para los circuitos de cuatro estados usando como elementos de memoria:

(a) *Flip-flops* T con reloj. **(c)** *Flip-flops* SR con reloj.

(b) *Flip-flops* JK con reloj.

	x = 0	x = 1
A	$C/0$	$D/0$
B	$C/0$	$A/0$
C	$B/0$	$D/0$
D	$A/1$	$B/1$

9.10 Deduzca las ecuaciones lógicas para implantar el circuito secuencial de cuatro estados definido mediante la siguiente tabla de estados, utilizando la asignación de estados dada y:

(a) *Flip-flops* D. **(c)** *Flip-flops* SR con reloj.

(b) *Flip-flops* JK con reloj. **(d)** *Flip-flops* T con reloj.

y_1	y_2
0	0
0	1
1	1
1	0

	x = 0	x = 1
A	$B/0$	$C/0$
B	$D/0$	$A/1$
C	$A/1$	$D/0$
D	$D/1$	$B/1$

9.11 Determine una asignación de estados para el siguiente circuito secuencial síncrono utilizando el procedimiento de asignación de estados presentado en este capítulo.

	x = 0	x = 1
A	$B/0$	$E/0$
B	$D/0$	$A/1$
C	$D/1$	$A/0$
D	$B/1$	$C/1$
E	$A/0$	$A/0$

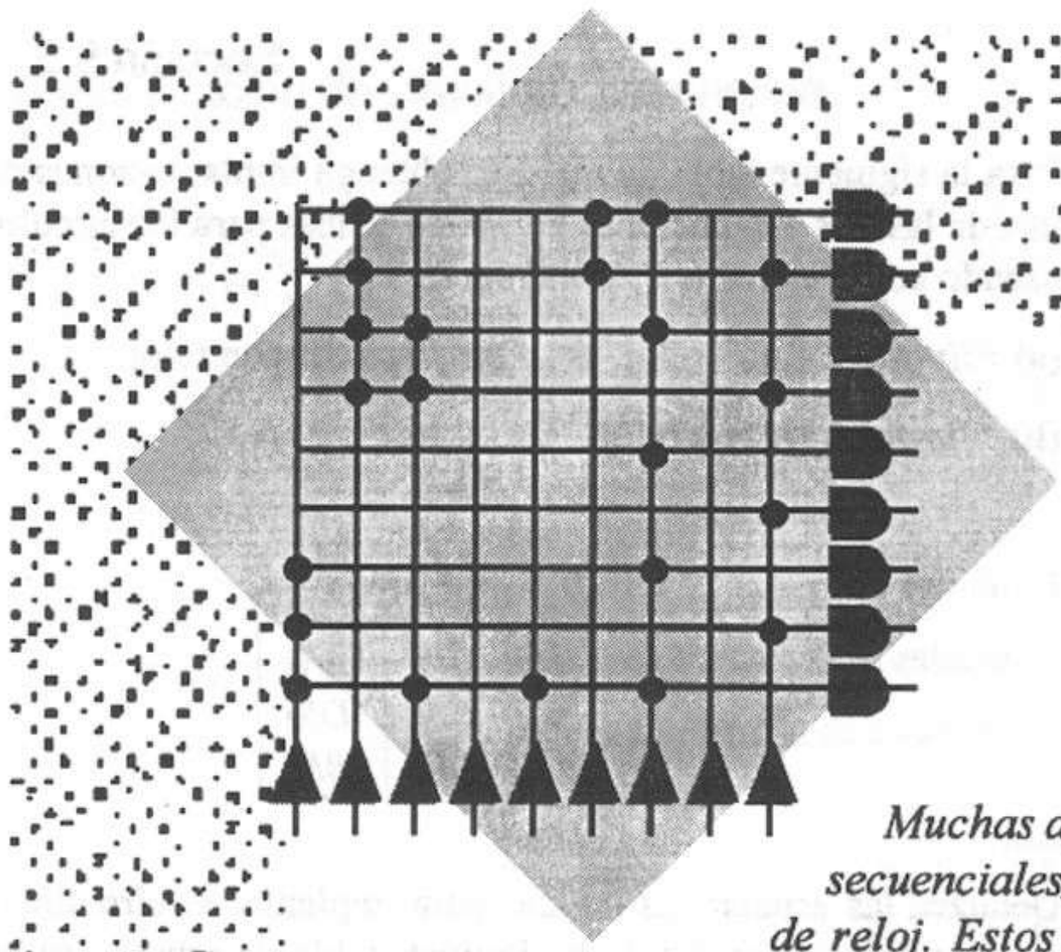
9.12 Determine una asignación de estados para el siguiente circuito secuencial. Elija como asignación para el estado A $y_3 = y_2 = y_1 = 0$.

	x = 0	x = 1
A	$B/0$	$E/0$
B	$A/1$	$C/1$
C	$B/0$	$C/1$
D	$C/0$	$E/0$
E	$D/1$	$A/0$

9.13 Determine una asignación de estados para el siguiente circuito.

	x = 0	x = 1
A	$B/0$	$D/1$
B	$A/1$	$C/0$
C	$D/0$	$A/0$
D	$C/1$	$B/1$

10



Muchas aplicaciones requieren el uso de circuitos secuenciales que no están sincronizados con una señal de reloj. Estos circuitos son los circuitos secuenciales asíncronos. Los circuitos asíncronos exigen una atención especial, pues no existe una señal de reloj que proporcione información común del tiempo a los elementos del circuito. Por tanto, los circuitos asíncronos responden de inmediato a cualquier cambio de entrada, en vez de responder a las entradas actuales durante un pulso de reloj. La ausencia de una señal de reloj también significa que las transiciones de los elementos de memoria deben iniciarse por otro medio, y es preciso tomar precauciones para evitar problemas relativos al tiempo. En este capítulo analizaremos las diversas facetas y ramificaciones del diseño de un circuito secuencial asíncrono.

Circuitos secuenciales asíncronos

10.1 Tipos de circuitos asíncronos

La primera clase de circuitos asíncronos que analizaremos es la de los *circuitos de modo pulso*. Estos circuitos tienen entradas de pulso y elementos de memoria sin reloj. Por lo general se utilizan *flip-flops* para la memoria en tales circuitos; por tanto, adoptaremos el modelo de la figura 10.1 como el modelo de circuito de modo pulso. Observe la gran semejanza con el modelo general de circuito secuencial dado en el capítulo 6. Sin embargo, existen ciertas restricciones sobre los circuitos de modo pulso que los hacen muy diferentes de los circuitos estudiados en el capítulo 8. Establecemos las siguientes hipótesis en el análisis y síntesis de los circuitos secuenciales asíncronos de modo pulso.

1. Nunca aparecen los pulsos en forma simultánea en dos o más líneas de entrada.
2. Las transiciones de los elementos de memoria sólo son iniciadas por los pulsos de entrada.
3. Las variables de entrada sólo se utilizan en la forma complementada o en la no complementada, pero no en ambas.

La primera hipótesis es muy práctica, pues la probabilidad de que dos pulsos ocurran exactamente al mismo tiempo es muy pequeña. Esta hipótesis es necesaria, ya que no se utiliza un reloj para sincronizar los cambios de estado. En una situación del mundo real, si intentamos aplicar dos pulsos de entrada a

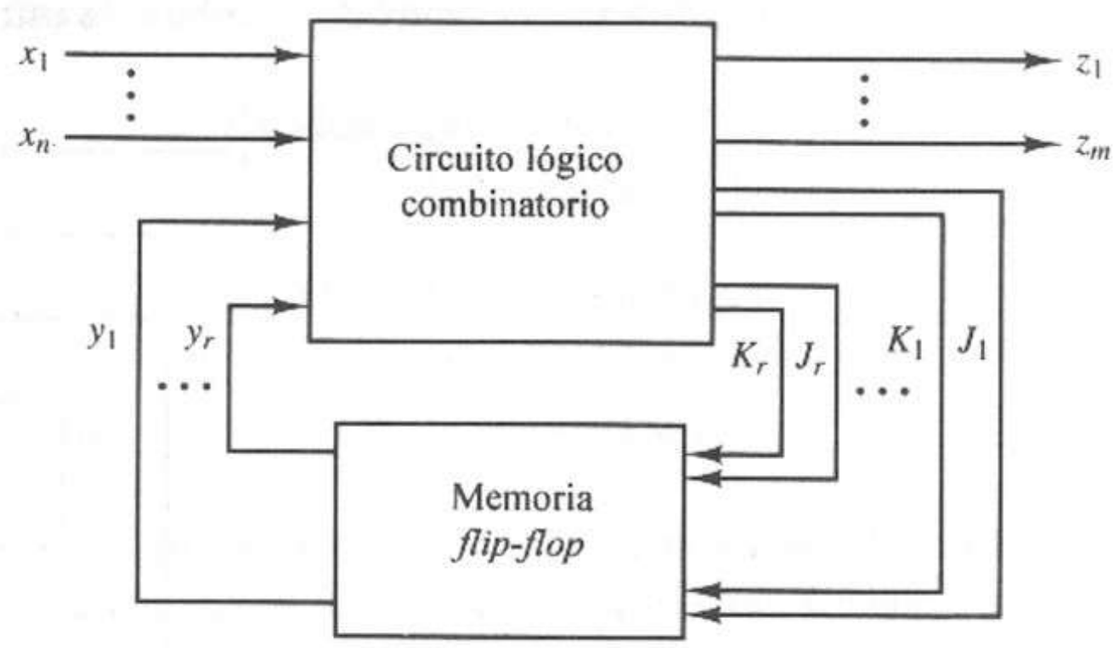


Figura 10.1 Modelo de circuito de modo pulso.

un circuito al mismo tiempo, digamos en las líneas de entrada x_1 y x_2, un pulso llegará antes que otro, y el funcionamiento del circuito dependería del pulso que llegó primero. Por tanto, el funcionamiento del circuito dependería de los valores de resistencia y de capacitancia parásitas del hardware. Esto no es aceptable como práctica de diseño, así que no permitimos diseños que acepten pulsos de entrada coincidentes. Además, la separación de los pulsos de entrada en el tiempo debe ser al menos igual al tiempo de respuesta del elemento de memoria más lento. Así, ningún elemento de memoria se encontrará en el proceso de cambio de estado cuando se presente un nuevo pulso de entrada. De este modo, el comportamiento del circuito será siempre predecible.

Consideremos ahora la información proporcionada por los pulsos de entrada al circuito. Se puede argumentar que como los pulsos pueden presentarse de manera asíncrona o, en otras palabras, en tiempos aleatorios desconocidos, el dispositivo sólo recibe información cuando se presenta un pulso. Por tanto, sólo se puede utilizar la forma no complementada de los pulsos de entrada para realizar los circuitos de modo pulso.

La segunda clase de circuitos secuenciales asíncronos por analizar está formada por los *circuitos de modo fundamental*. Estos circuitos tienen entradas de nivel y elementos de memoria sin reloj. La figura 10.2 muestra el modelo de un circuito de modo fundamental. Los elementos de memoria aparecen como líneas de retardo en el modelo. Sin embargo, en la práctica, por lo general no es necesario utilizar una línea de retardo física, ya que existe un retardo suficiente en los otros elementos del circuito lógico. Para esta presentación, supondremos que todos los retardos en el circuito se pueden agrupar en los elementos de retardo mostrados en las rutas de retroalimentación. También supondremos que cada elemento de retardo tiene la misma magnitud de retardo Δt. Estas dos últimas hipótesis no siempre se pueden justificar en la práctica y serán eliminadas en un punto posterior del capítulo.

Para el funcionamiento en modo fundamental, las entradas están restringidas de modo que sólo se permite que una variable de entrada cambie su valor en un instante dado. Esta restricción, similar a la requerida en los circuitos de modo pulso, es real, pues en las situaciones prácticas dos o más entradas no cambian de valor precisamente en el mismo instante. Por tanto, el segundo cambio y los posteriores se podrían presentar mientras el circuito se encuentra aún respondiendo al primer cambio de entrada, en cuyo caso sería posible un comportamiento incorrecto del circuito. Ocurre una situación similar si el tiempo entre dos cambios de entrada es muy pequeño. Para un funcionamiento predecible

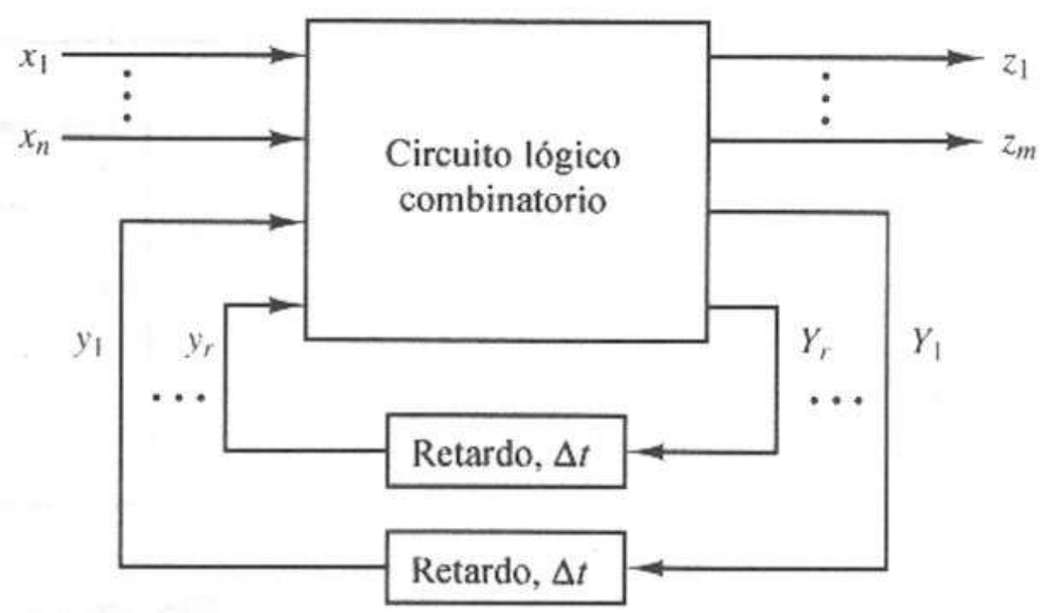


Figura 10.2 Modelo de circuito de modo fundamental.

de los circuitos de modo fundamental hay que espaciar los cambios de entrada en al menos un tiempo Δt, el tiempo necesario para que el circuito se establezca en un estado siguiente estable después de un cambio de entrada.

10.2 Análisis de circuitos asíncronos de modo pulso

En el análisis de los circuitos secuenciales asíncronos de modo pulso es importante recordar que estos circuitos responden de inmediato a los pulsos en sus entradas en vez de esperar una señal de reloj como los circuitos secuenciales síncronos. La ausencia de una señal de reloj también significa que las transiciones de elementos de memoria deben iniciarse por algún otro medio. Por tanto, hay que tomar precauciones para evitar problemas de tiempos, poniendo mucha atención y entendiendo el significado de las tres hipótesis relativas al funcionamiento en modo pulso presentadas en la sección anterior.

La primera hipótesis, en el sentido de que los pulsos no ocurren de manera simultánea en dos o más líneas de entrada, significa que un circuito con n líneas de entrada sólo tiene $n+1$ condiciones de entrada, en vez de 2^n, como es el caso de los circuitos síncronos. La hipótesis 2 implica que una transición de estado sólo se puede presentar si ocurre un pulso de entrada. Por tanto, los elementos de memoria del circuito sólo responden cuando llega un pulso de entrada. La tercera hipótesis garantiza que todos los dispositivos se disparan en la misma transición de cada pulso. Es importante tener en mente estas hipótesis al analizar los siguientes ejemplos.

EJEMPLO 10.1

Examinar el comportamiento del circuito asíncrono con pulso que se muestra en la figura 10.3.

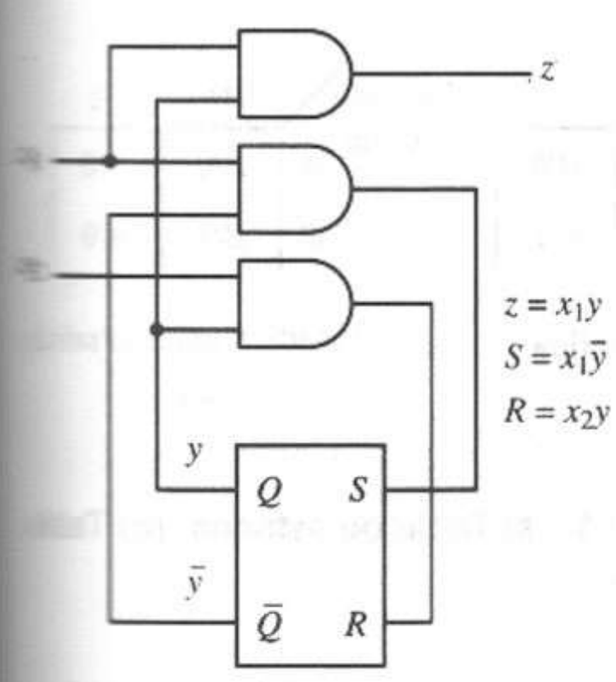


Figura 10.3 Circuito secuencial asíncrono con pulso.

El análisis de este circuito es muy similar al de un circuito síncrono. Las principales diferencias se deben a la ausencia de una señal de reloj y a las hipótesis ya enumeradas.

El circuito tiene los siguientes estados y entradas:

$$\text{Estados:}\quad [y] = 0 \equiv A$$
$$[y] = 1 \equiv B$$

$$\text{Entradas:}\quad [x_1,x_2] = 00 \equiv I_0$$
$$[x_1,x_2] = 10 \equiv I_1$$
$$[x_1,x_2] = 01 \equiv I_2$$

La figura 10.4 muestra un diagrama de tiempos del circuito para una secuencia de entrada típica. Observe que todas las transiciones de estado corresponden a la ocurrencia de un pulso de entrada. Las señales S y R se activan en la transición inicial del pulso de entrada. El *flip-flop* responde a sus entradas S y R en τ segundos. La salida cambiante del *flip-flop* y desactiva entonces las señales S y R, y entonces el circuito espera otro pulso de entrada en x_1 o x_2.

Podemos construir la tabla de estados de la figura 10.5a a partir del diagrama de tiempos de la figura 10.4. La figura 10.5b muestra una tabla de estados simbólica, con las tres combinaciones de x_1x_2 designadas como condiciones de entrada I_0, I_1 e I_2. Observe que no existe una columna para la condición $x_1 = x_2 = 1$, ya que hemos supuesto que no pueden ocurrir pulsos simultáneos sobre varias entradas.

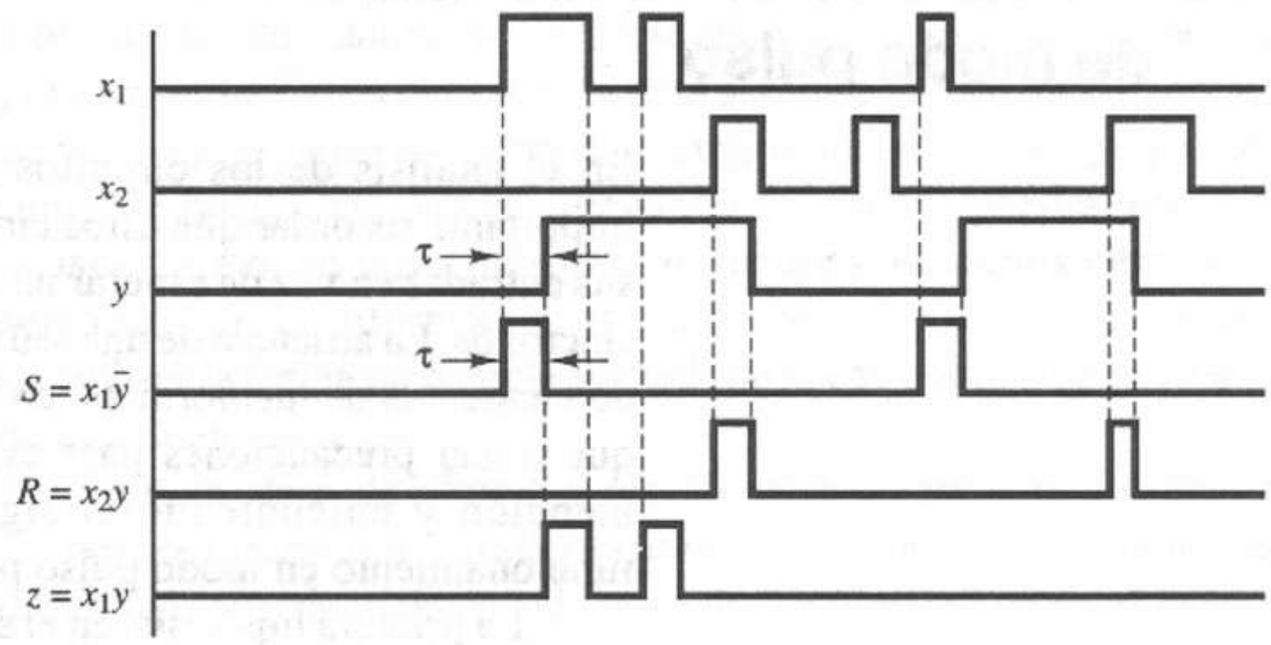


Figura 10.4 Diagrama de tiempos para el circuito del ejemplo.

La figura 10.5c muestra una tabla de estados simplificada. Obtenemos esta tabla a partir de una tabla simbólica por los siguientes pasos:

1. Eliminamos la columna *I*0 correspondiente a la condición de entrada $x_1 = x_2 = 0$. Como no hay cambio cuando no existe un pulso sobre alguna entrada, esta columna no proporciona información significativa.

2. Intercambiamos las columnas I_2 e I_1 y remplazamos los símbolos I_1 con x_1 e I_2 por x_2, indicando los pulsos sobre x_1 y x_2, respectivamente.

La tabla de estados simplificada describe por completo el comportamiento del circuito.

y \ x_1x_2	00	01	10
0	0/0	0/0	1/0
1	1/0	0/0	1/1

Y/z

(a)

Estado actual	I_0	I_2	I_1
A	$A/0$	$A/0$	$B/0$
B	$B/0$	$A/0$	$B/1$

Estado siguiente/salida

(b)

Estado actual	x_1	x_2
A	$B/0$	$A/0$
B	$B/1$	$A/0$

Estado siguiente/salida

(c)

Figura 10.5 Tablas de estados para el ejemplo 10.1. **(a)** Tabla de estados. **(b)** Tabla de estados simbólica. **(c)** Tabla de estados simplificada.

La figura 10.6 muestra la obtención del mapa K de la tabla de estados. Como hemos supuesto que la condición de entrada $x_1x_2 = 11$ no ocurre, las celdas correspondientes del mapa K quedan sin especificar. En el paso final eliminamos

las columnas 00 y 11 de la tabla y reetiquetamos la columna 10 como x_1 y la columna 01 como x_2.

y \ x_1x_2	00	01	11	10
0	**0**	**0**	–	**1**
1	**0**	**0**	–	**0**

S

y \ x_1x_2	00	01	11	10
0	**0**	**0**	–	**0**
1	**0**	**1**	–	**0**

R

y \ x_1x_2	00	01	11	10
0	**0**	**0**	–	**0**
1	**0**	**0**	–	**1**

z

(a)

y \ x_1x_2	00	01	11	10
0	**00**	**00**	–	**10**
1	**00**	**01**	–	**00**

SR

y \ x_1x_2	00	01	11	10
0	**0**	**0**	–	**1**
1	**1**	**0**	–	**1**

Y

(b)

y \ x_1x_2	00	01	11	10
0	**0/0**	**0/0**	—	**1/0**
1	**1/0**	**0/0**	—	**1/1**

Y/z

y	x_1	x_2
0	**1/0**	**0/0**
1	**1/1**	**0/0**

Y/z

(c)

Figura 10.6 Desarrollo del mapa K de las tablas de estados. **(a)** Mapas K para el circuito. **(b)** Mapa SR y mapa de estado siguiente (Y) combinados. **(c)** Tablas de estados.

Analizamos los circuitos asíncronos con pulso que utilizan elementos de memoria no disparados por transición (es decir, *latches*) como lo muestra el ejemplo anterior. A continuación analizaremos un circuito que utiliza *flip-flops* disparados por transición.

EJEMPLO 10.2

Analizar el circuito secuencial de la figura 10.7 mediante un diagrama de tiempos. El circuito queda descrito por las ecuaciones lógicas

$$D_1 = \bar{y}_1, \quad D_2 = \bar{y}_1, \quad z = xy_1y_2$$
$$C_1 = xy_2, \quad C_2 = x$$

En la figura 10.8 hemos construido el diagrama de tiempos para el circuito. La entrada x está compuesta por pulsos asíncronos, y el estado inicial del circuito es $y_1 = y_2 = 0$. Observe que sólo aparecen tres estados del circuito.

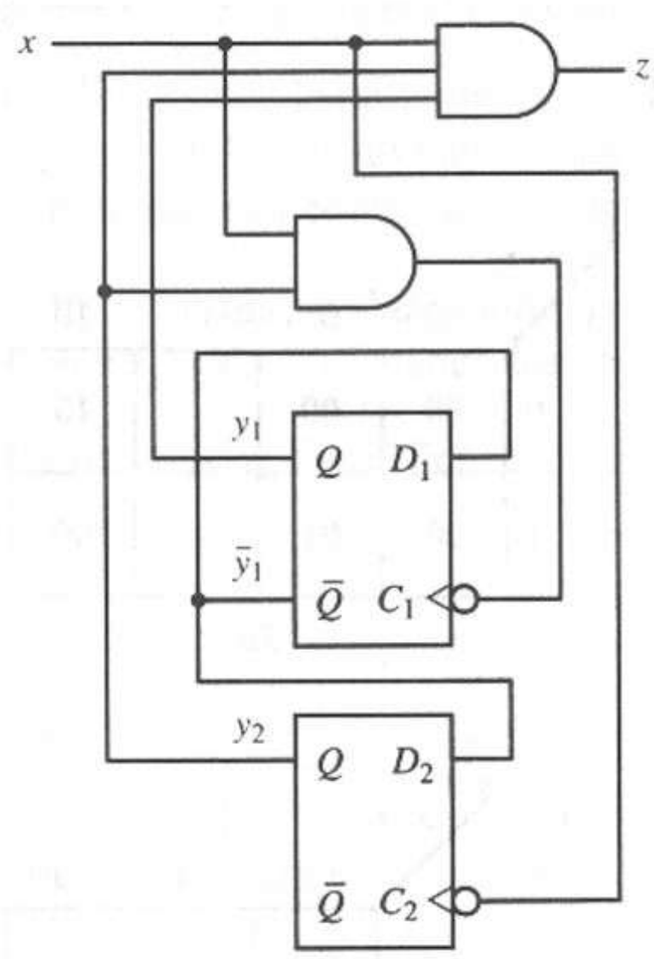


Figura 10.7 Circuito secuencial con pulso y memoria sin reloj.

Si el estado inicial hubiera sido $y_1 = 1$ y $y_2 = 0$, el circuito secuencial no podría cambiar de estado, ya que

$$D_1 = 0, D_2 = 0$$
$$C_1 = 0, C_2 = x$$

La variable de estado y_2 permanecería en 0 e inhibiría cualquier cambio de estado en la salida del *flip-flop* y_2.

Podemos compilar una tabla y un diagrama de estados para este circuito si definimos lo siguiente:

Entradas: $I_0 \equiv$ ningún pulso en x
$I_1 \equiv$ pulso en x

Estados: $\underline{y_1 y_2}$
$A \equiv 00$
$B \equiv 01$
$C \equiv 10$
$D \equiv 11$

Salidas: $z = 0$
$z = 1$

La figura 10.9 muestra la tabla y el diagrama de estados resultantes, los cuales se deducen a partir del diagrama de tiempos de la figura 10.8. Observe que el diagrama de estados tiene dos partes independientes, así que si el estado

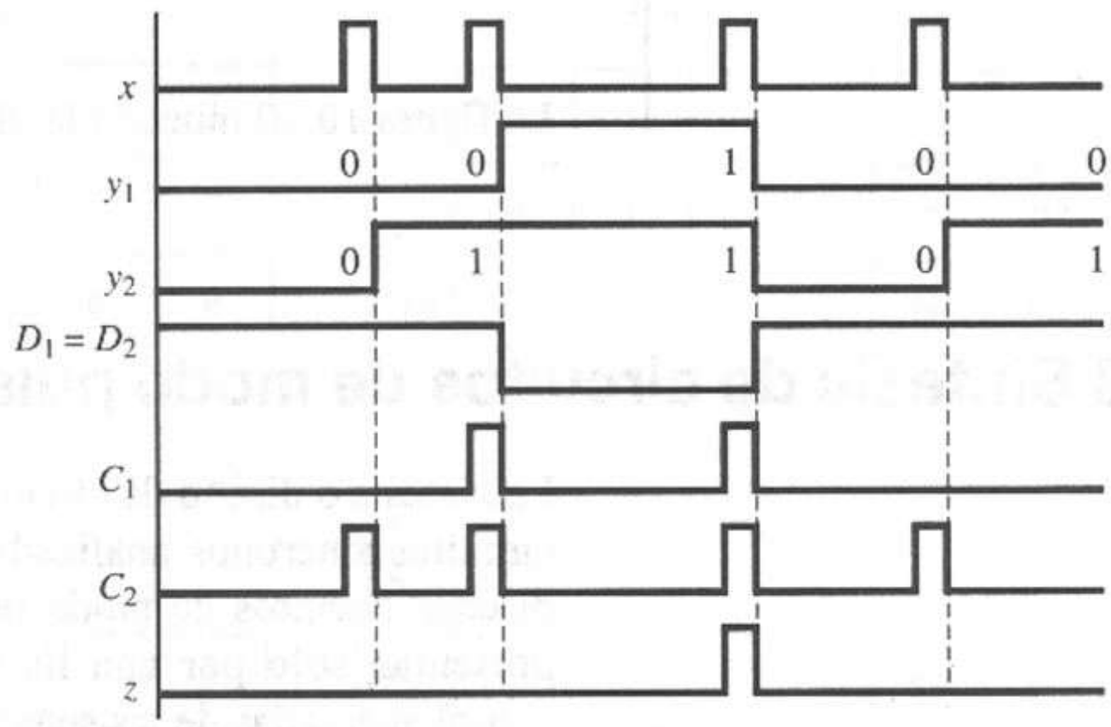
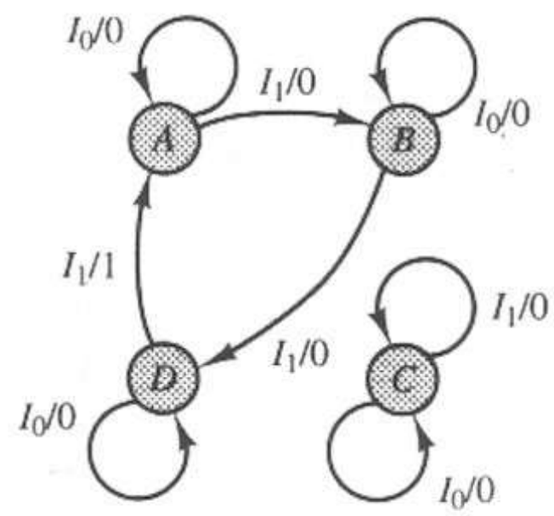


Figura 10.8 Diagrama de tiempos para el ejemplo 10,2

Estado actual	I_0	I_1
A	A/0	B/0
B	B/0	D/0
C	C/0	C/0
D	D/0	A/1

Estado siguiente/salida

(a)



(b)

Figura 10.9 Tabla y diagrama de estados del ejemplo 10.2. **(a)** Tabla de estados. **(b)** Diagrama de estados.

C es el estado inicial, el circuito secuencial queda "colgado" ahí. Sin embargo, si el estado inicial es A, B, o D, el circuito se comporta como una máquina secuencial típica.

Podemos utilizar mapas K para construir la tabla de estados correspondiente al circuito de la figura 10.7, con base en la ecuación característica del *flip-flop* D (ecuación 6.12) y la siguiente observación. Las entradas de reloj del *flip-flop* D verán una transición 1 o 0 sólo si se presenta un pulso de entrada. Por tanto, sólo se puede presentar como máximo una de tales transiciones para cada pulso de entrada.

Obtenemos las siguientes ecuaciones a partir de la ecuación 6.12 y las ecuaciones lógicas para el circuito de la figura 10.7:

$$\begin{aligned} Y_1 &= D_1 C_1 + y_1 \bar{C}_1 \\ &= \bar{y}_1 x y_2 + y_1(\bar{x} + \bar{y}_2) \\ &= x\bar{y}_1 y_2 + \bar{x} y_1 + y_1 \bar{y}_2 \\ Y_2 &= D_2 C_2 + y_2 \bar{C}_2 \\ &= \bar{y}_1 x + y_2 \bar{x} \end{aligned}$$

La figura 10.10 muestra la obtención del mapa K de la tabla de estados.

10.3 Síntesis de circuitos de modo pulso

La síntesis o diseño de circuitos de modo pulso es muy similar al diseño de los circuitos síncronos analizado en el capítulo 8. Sin embargo, recuerde que al diseñar circuitos de modo pulso, no existe un pulso de reloj, las entradas se presentan sólo por una línea a la vez y sólo se pueden utilizar formas no complementadas de las señales de entrada.

La ausencia de pulsos de reloj implica que el disparo del *latch* o *flip-flop* debe realizarse mediante los pulsos en las señales de entrada y, por tanto, hay que obtener toda la información de tiempo del circuito a partir de los pulsos de entrada. Entonces, los pulsos de entrada no sólo proporcionan la información de entrada sino que también asumen las funciones realizadas por el pulso de reloj en los circuitos síncronos.

10.3.1 Procedimiento de diseño para circuitos de modo pulso

El procedimiento de diseño por pasos que presentamos a continuación es similar al de los circuitos síncronos. Sin embargo, los detalles de algunos pasos son diferentes, como veremos en los tres ejemplos que siguen.

Paso 1. Deducir un diagrama o tabla de estados.

Paso 2. Minimizar la tabla de estados.

Paso 3. Elegir una asignación de estados y generar la tabla de transición/salida.

Paso 4. Seleccionar el tipo de *latch* o *flip-flop* por utilizar y determinar las ecuaciones de excitación.

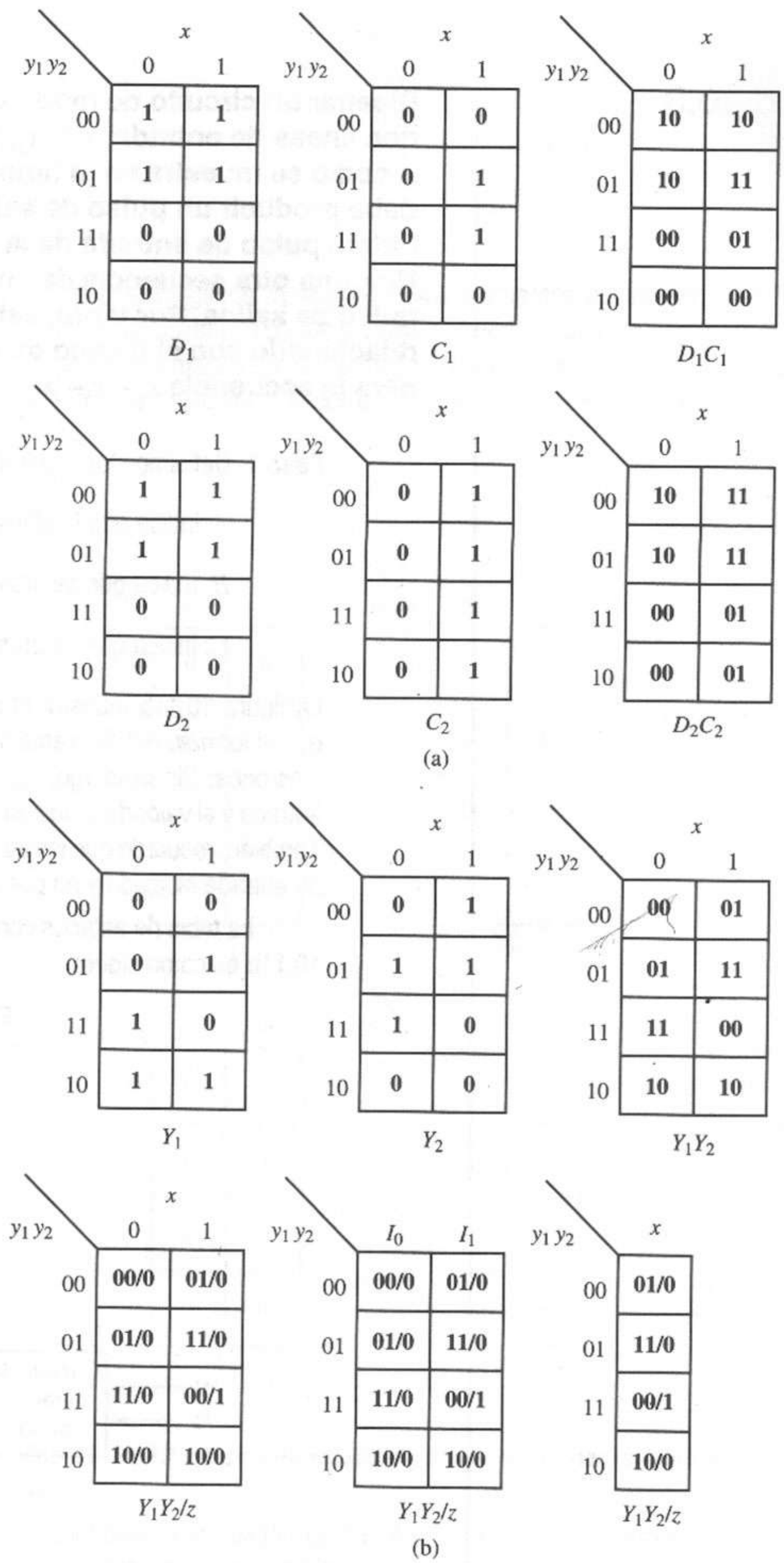


Figura 10.10 Desarrollo de una tabla de estados mediante mapas K. **(a)** Mapas de entrada de *flip-flop*. **(b)** Mapas de estado siguiente.

Paso 5. Determinar las ecuaciones de salida.

Paso 6. Elegir los elementos lógicos adecuados y trazar el diagrama del circuito.

EJEMPLO 10.3

Diseñar un circuito de modo pulso con dos líneas de entrada, x_1 y x_2, y una línea de salida, z, como se muestra en la figura 10.11a. El circuito debe producir un pulso de salida que coincida con el último pulso de entrada de la secuencia x_1—x_2—x_2. Ninguna otra secuencia de entrada debe producir un pulso de salida. Por tanto, este ejemplo está relacionado con el diseño de un detector de secuencia, para la secuencia x_1—x_2—x_2.

Paso 1. Definimos los siguientes tres estados del circuito:

A: indica que la última entrada fue x_1.

B: indica que se presentó la secuencia x_1—x_2.

C: indica que se presentó la secuencia x_1—x_2—x_2.

La figura 10.11b muestra el diagrama de estados correspondiente. Observe que el formato del diagrama de estados es similar al utilizado para los circuitos síncronos. Sin embargo, las transiciones están rotuladas con la variable de entrada y el valor de salida en vez de utilizar ambos valores de entrada y salida. También, recuerde que las transiciones de estado se disparan al ocurrir el pulso de entrada indicado y no por un pulso de reloj.

La tabla de estados correspondiente al diagrama de estados de la figura 10.11b es como sigue:

Estado actual	x_1	x_2
A	$A/0$	$B/0$
B	$A/0$	$C/1$
C	$A/0$	$C/0$

Estado siguiente/salida

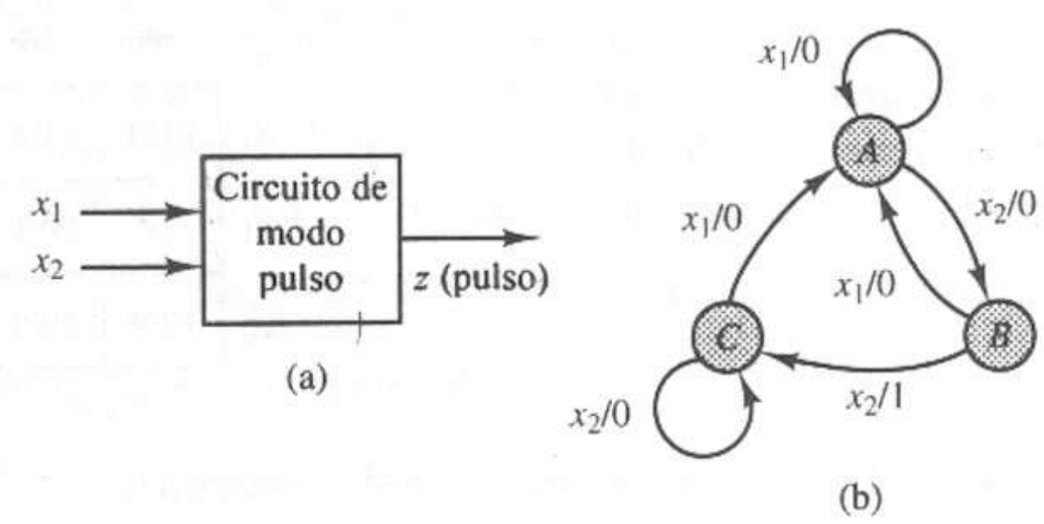


Figura 10.11 Ejemplo de modo pulso. **(a)** Circuito de modo pulso. **(b)** Diagrama de estados.

Paso 2. La tabla de estados ya es mínima.

Paso 3. La asignación de estados $A = 00$, $B = 01$, $C = 10$ produce la siguiente tabla de transición/salida.

y_1y_2	x_1	x_2
00	00/0	01/0
01	00/0	10/1
10	00/0	10/0

Y_1Y_2/z

Paso 4. Elegimos *flip-flops* T como elementos de memoria. La figura 10.12 muestra los mapas de estado siguiente y de excitación correspondientes al *flip-flop*. Estos mapas son como mapas de Karnaugh de cuatro variables, reducidos. Omitimos las columnas correspondientes a $x_1 = x_2 = 0$ y $x_1 = x_2 = 1$ ya que no

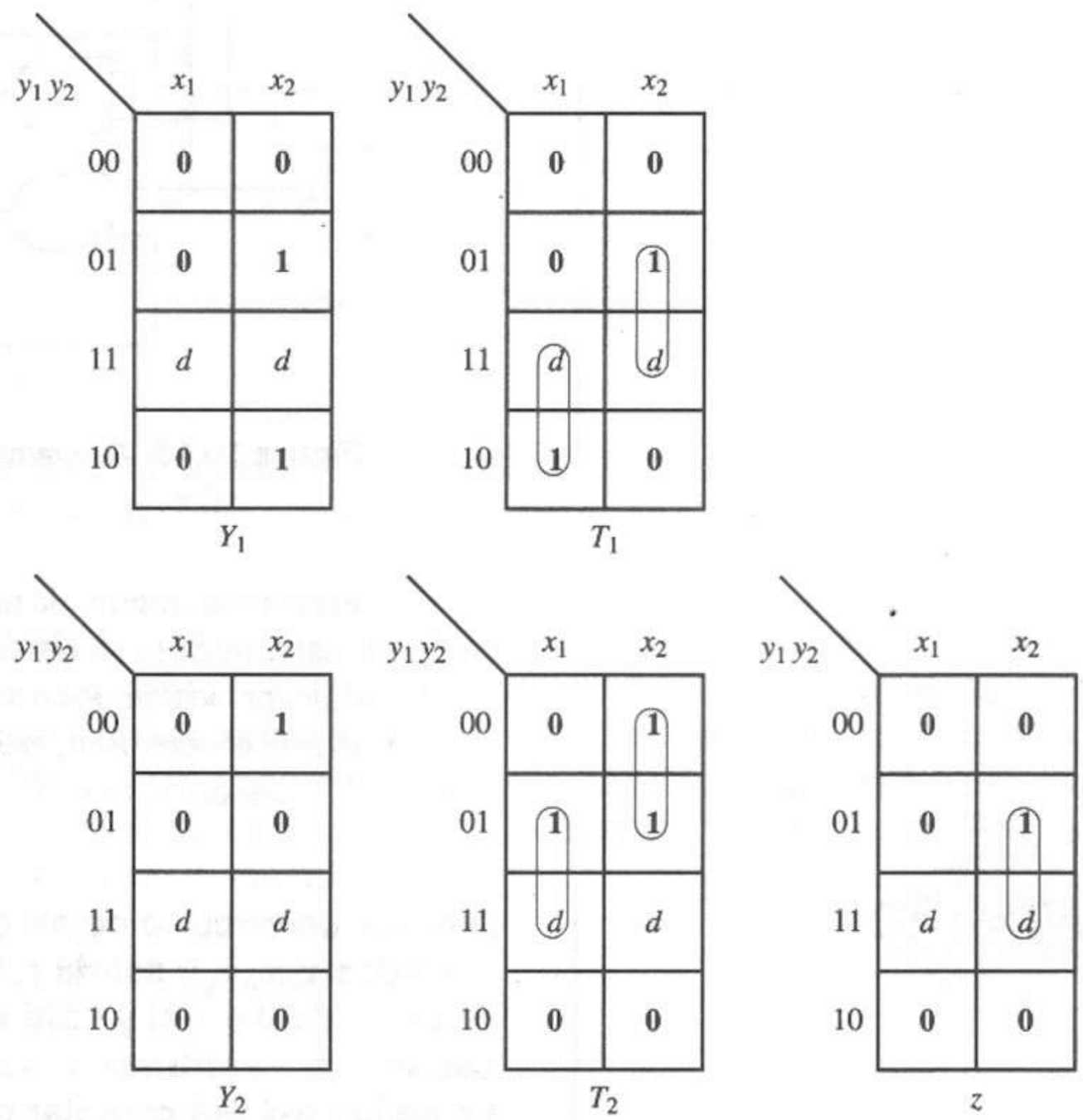


Figura 10.12 Mapas de estado siguiente, de excitación y de salida.

contienen información pertinente. El lector debe verificar que las columnas omitidas no son necesarias. Como las columnas restantes no son adyacentes en el mapa completo, sólo podemos agrupar dentro de una columna dada. Por tanto, obtenemos las siguientes ecuaciones de excitación:

$$T_1 = x_1 y_1 + x_2 y_2$$
$$T_2 = x_1 y_2 + x_2 \bar{y}_1$$

Paso 5. La figura 10.12 muestra el mapa de salida, del cual deducimos la siguiente ecuación:

$$z = x_2 y_2$$

Paso 6. Podemos utilizar la lógica AND/OR para realizar las ecuaciones. La figura 10.13 muestra el circuito resultante.

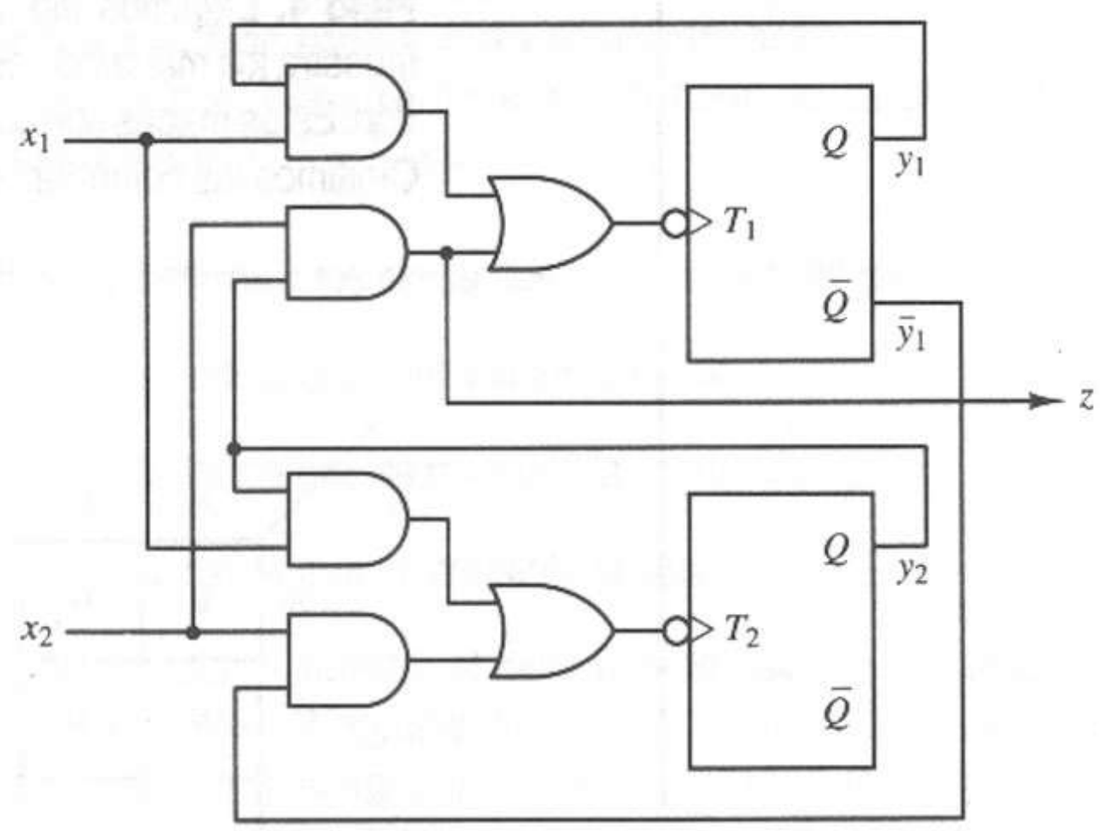


Figura 10.13 Diagrama lógico para el ejemplo de modo pulso.

En el ejemplo anterior, la realización del circuito tomó la forma de un circuito de tipo Mealy, pues la salida fue función de una entrada y de una variable de estado. Ahora presentamos un segundo ejemplo que describe la realización de un circuito de tipo Moore. Recuerde que en el capítulo 8 definimos los circuitos de tipo Mealy y Moore.

EJEMPLO 10.4

Diseñar un circuito de modo pulso con entradas x_1, x_2, x_3 y salida z. La salida debe cambiar de 0 a 1 si y sólo si ocurre la secuencia de entrada x_1—x_2—x_3 mientras $z = 0$. La salida deberá cambiar de 1 a 0 sólo después de que se presente una entrada x_2.

Paso 1. Como la salida debe permanecer alta entre los pulsos de entrada, necesitamos un circuito tipo Moore para realizar la red de la figura 10.14a. El diagrama y la tabla de estados de las figuras 10.14b y c, respectivamente, satisfacen las condiciones establecidas.

Paso 2. Reducimos la tabla de la figura 10.14c.

Paso 3. La asignación de estados $A = 00$, $B = 01$, $C = 11$, $D = 10$ implica la tabla de transición/salida de la figura 10.14d.

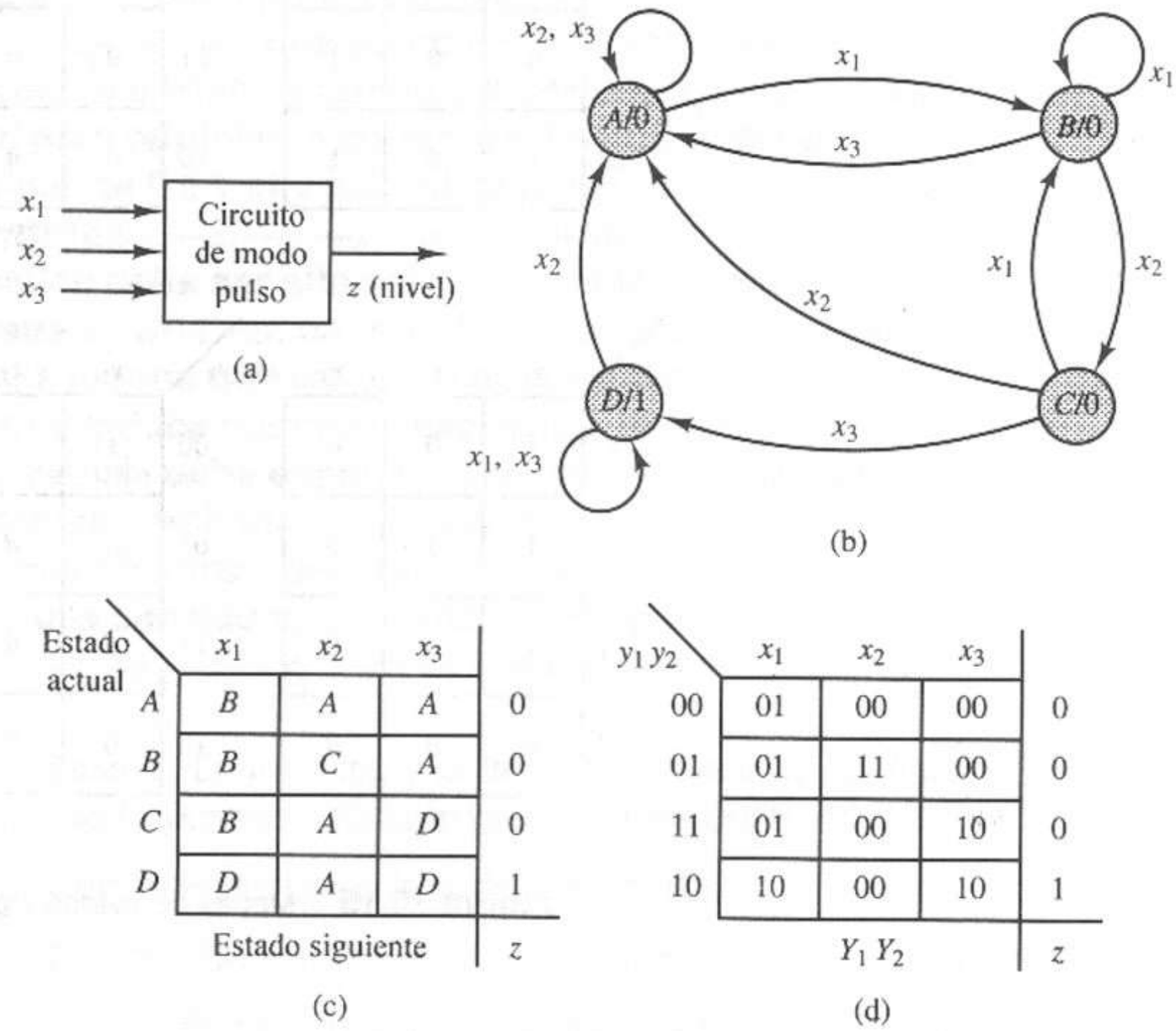


Figura 10.14 Ejemplo de modo pulso. **(a)** Circuito de modo pulso. **(b)** Diagrama de estados. **(c)** Tabla de estados. **(d)** Tabla de transición/salida.

Paso 4. La figura 10.15 muestra los mapas de estado siguiente y de excitación correspondientes a los *latches* SR. Observe que para utilizar *latches* SR, cada pulso de entrada debe durar lo suficiente como para iniciar un cambio de estado. Además, recuerde que al utilizar los mapas K reducidos hay que restringir los agrupamientos dentro de una columna dada. Los mapas producen las siguientes ecuaciones de excitación:

$$S_1 = x_2\bar{y}_1 y_2, \qquad R_1 = x_1 y_2 + x_2 y_1$$
$$S_2 = x_1\bar{y}_1, \qquad R_2 = x_2 y_1 + x_3$$

Paso 5 Como estamos realizando un circuito de tipo Moore, z sólo depende de las variables de estado. Obtenemos una salida 1 sólo cuando el circuito está en el estado D. Por tanto,

$$z = y_1\bar{y}_2$$

Paso 6. La figura 10.16 muestra el diagrama de circuito obtenido al utilizar la lógica AND/OR para realizar las ecuaciones.

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	0	0	0	0
01	0	1	0	0
11	0	0	1	0
10	1	0	1	0

Y_1

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	0	0	0	0
01	0	1	0	0
11	0	0	d	0
10	d	0	d	0

S_1

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	d	d	d	d
01	d	0	d	d
11	1	1	0	0
10	0	1	0	0

R_1

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	1	0	0	0
01	1	1	0	0
11	1	0	0	0
10	0	0	0	0

Y_2

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	1	0	0	0
01	d	d	0	0
11	d	0	0	0
10	0	0	0	0

S_2

$y_1 y_2$ \ $x_1 x_2 x_3$	000	001	011	010
00	0	d	d	d
01	0	0	1	1
11	0	1	1	1
10	d	d	d	d

R_2

Figura 10.15 Mapas de estado siguiente y de excitación de SR.

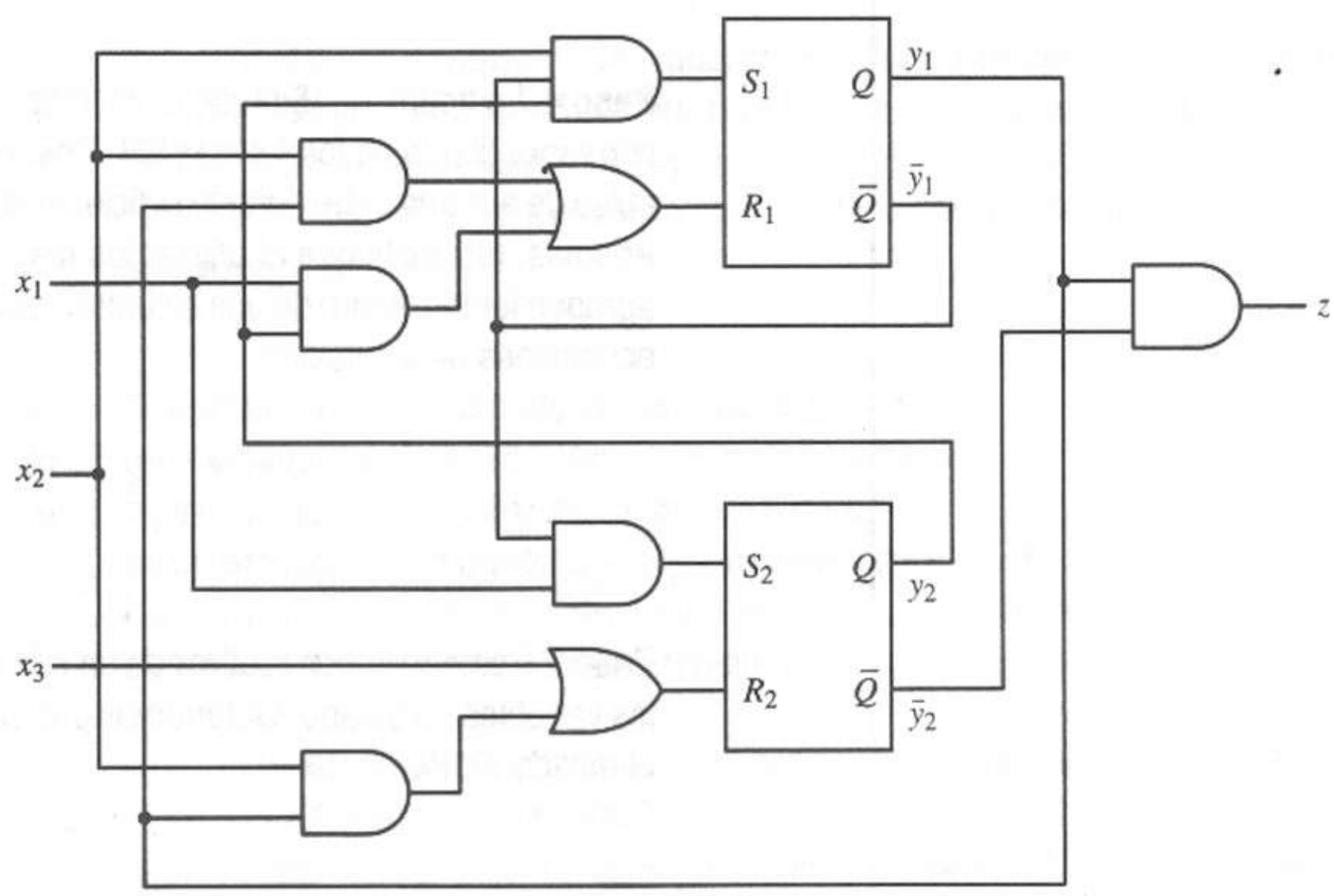


Figura 10.16 Diagrama de circuito para el ejemplo de modo pulso.

EJEMPLO 10.5

Diseñar un circuito de modo pulso con entradas x_1, x_2, x_3 y salida z. El circuito se utilizará como cerradura digital, como muestra la figura 10.14a, con la combinación $x_1—x_2—x_2—x_1—x_2$. Los pulsos en las entradas ocurren al oprimir botones. La entrada x_3 es una señal *reset* para reiniciar la cerradura. La salida debe cambiar de 0 a 1 si y sólo si se presenta la secuencia de entrada $x_1—x_2—x_2—x_1—x_2$ mientras $z = 0$. La salida debe ser alta sólo durante la última entrada x_2. Una vez dada x_2, la cerradura no se puede abrir, a menos que un pulso en x_3 reinicie la lógica. Si se introduce cualquier secuencia incorrecta, la cerradura debe esperar un pulso *reset* antes de aceptar la combinación correcta. En otras palabras, después de que se ha abierto la cerradura, necesita la secuencia $x_3—x_1—x_2—x_2—x_1—x_2$ para abrirse de nuevo.

Paso 1. El diagrama y la tabla de estados de las figuras 10.17b y c, respectivamente, satisfacen las condiciones establecidas.

Paso 2. Reducimos la tabla de la figura 10.17c.

Paso 3. Utilizamos las reglas de asignación de estados del capítulo 9:

Regla 1: Trataremos de satisfacer las siguientes adyacencias:

$$\begin{array}{llllll} BC, & BE, & BF, & CE, & CF, & EF \\ AD, & AE, & AF, & DE, & DF & EF \end{array}$$

Regla 3: Establecemos una gráfica de implicación como en la figura 10.17d. De la subgráfica cerrada, encerrada en un círculo dentro de la gráfica, intentamos satisfacer las siguientes adyacencias:

$$CD, \quad DF, \quad EF, \quad CE, \quad BC$$

La asignación de estados de la figura 10.17e es una buena opción para este circuito. Utilizamos esta asignación de estados para obtener la tabla de transición de la figura 10.17f.

Paso 4. A partir de la tabla de transición, deducimos los mapas de excitación de los *latches* SR, dados en la figura 10.17g. Recuerde que al utilizar los mapas K reducidos los agrupamientos sólo deben hacerse dentro de una columna dada. Obtenemos las siguientes ecuaciones de excitación a partir de los mapas.

$$S_1 = y_2x_1 + x_2$$
$$R_1 = x_3$$

$$S_2 = \bar{y}_1\bar{y}_2x_1$$
$$R_2 = y_2x_1 + y_3x_2 + x_3$$

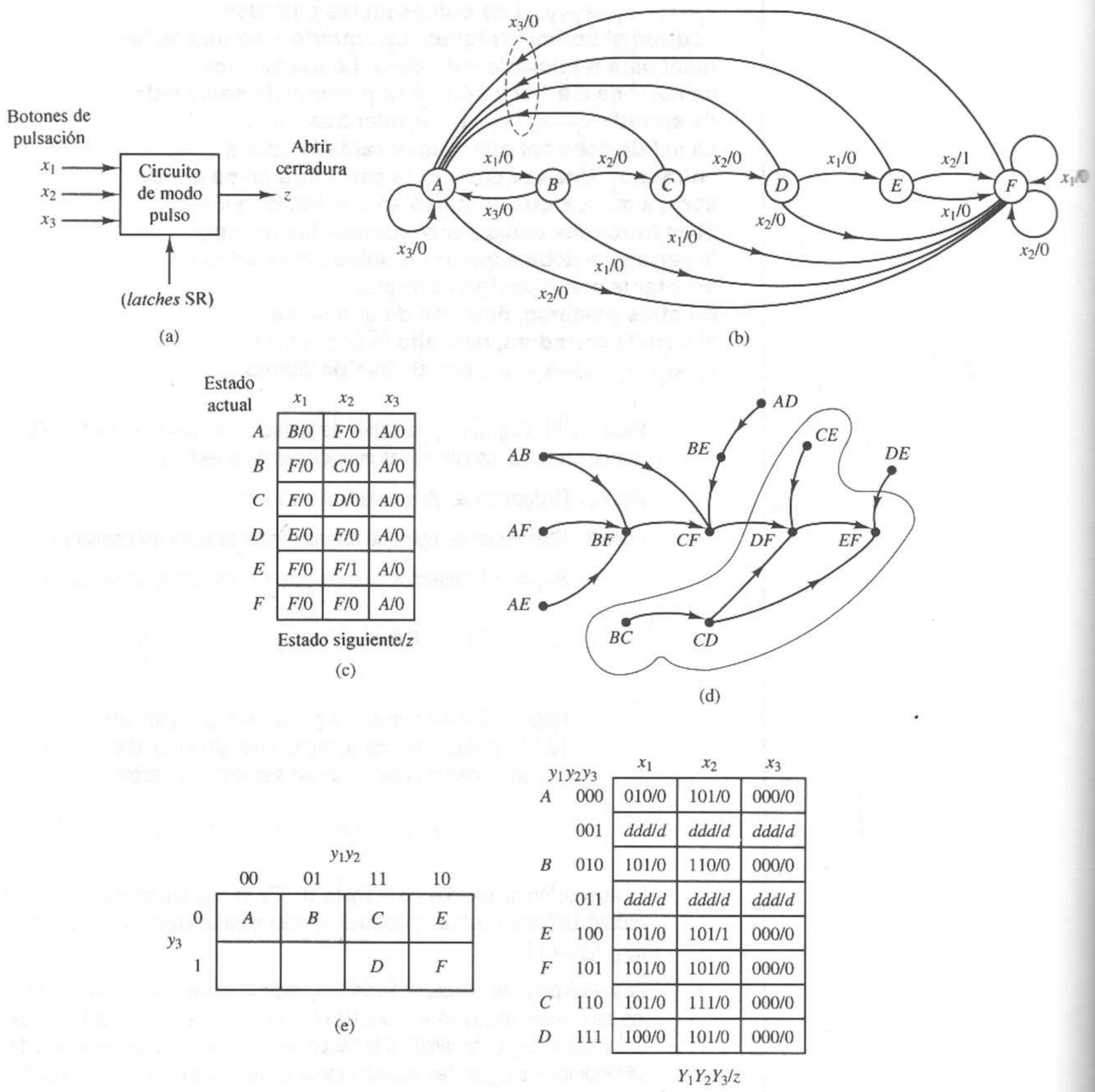


Figura 10.17 Ejemplo de cerradura de combinación digital. **(a)** Cerradura de combinación digital. **(b)** Diagrama de estados. **(c)** Tabla de estados. **(d)** Gráfica de implicación. **(e)** Asignación de estados. **(f)** Tabla de transición/salida.

$y_1y_2y_3$	x_1	x_2	x_3
000	**0**d	**10**	**0**d
001	dd	dd	dd
010	**10**	**10**	**0**d
011	dd	dd	dd
100	d**0**	d**0**	**01**
101	d**0**	d**0**	**01**
110	d**0**	d**0**	**01**
111	d**0**	d**0**	**01**

S_1R_1

$y_1y_2y_3$	x_1	x_2	x_3
000	**10**	**0**d	**0**d
001	dd	dd	dd
010	**01**	d**0**	**01**
011	dd	dd	dd
100	**0**d	**0**d	**0**d
101	**0**d	**0**d	**0**d
110	**01**	d**0**	**01**
111	**01**	**01**	**01**

S_2R_2

$y_1y_2y_3$	x_1	x_2	x_3
000	**0**d	**10**	**0**d
001	dd	dd	dd
010	**10**	**0**d	**0**d
011	dd	dd	dd
100	**10**	**10**	**0**d
101	d**0**	d**0**	**01**
110	**10**	**10**	**0**d
111	**01**	d**0**	**01**

S_3R_3

(g)

y_3 \ y_1y_2	00	01	11	10
0	**0**	**0**	**0**	**1**
1	d	d	**0**	**0**

(h)

Figura 10.17 (continuación) Ejemplo de cerradura de combinación digital. **(g)** Mapas K de excitación. **(h)** Mapa K de salida.

$$S_3 = (y_2\bar{y}_3 + y_1\bar{y}_3)x_1 + (\bar{y}_2 + y_1)x_2$$
$$R_3 = (y_2y_3)x_1 + x_3$$

Paso 5. La figura 10.17h muestra el mapa K para la salida z. La ecuación de salida es

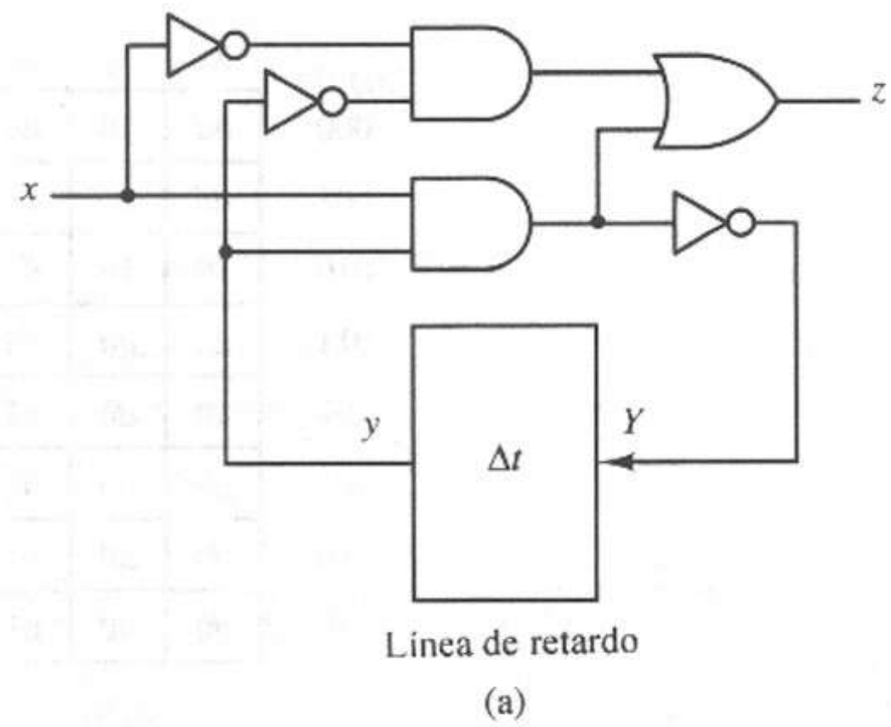
$$z = y_1\bar{y}_2\bar{y}_3x_2$$

Paso 6. La construcción detallada del diagrama de circuito queda como ejercicio para el lector.

Con este ejemplo concluye nuestro análisis de los circuitos de modo pulso. Dedicaremos el resto del capítulo a los circuitos de modo fundamental.

10.4 Análisis de circuitos de modo fundamental

El análisis de un circuito de modo fundamental debe ser muy cuidadoso, pues este tipo de circuitos tiene características de comportamiento muy especial; es decir, estos circuitos utilizan memoria sin reloj y entradas de nivel, sincronizados mediante una señal de reloj o sin ésta. Por ejemplo, examinemos el circuito definido en la figura 10.18a. Esta red está compuesta por compuertas AND, OR y NOT con un elemento de memoria de línea de retardo. Dicho elemento puede



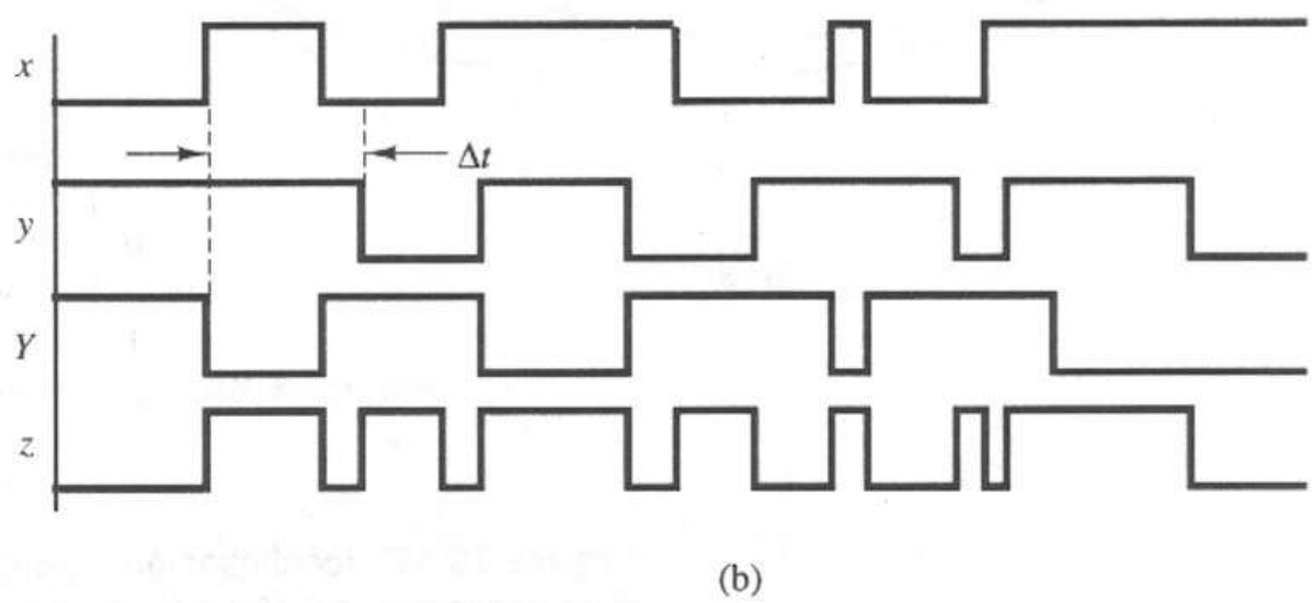


Figura 10.18 Circuito secuencial asíncrono de nivel. **(a)** Diagrama lógico. **(b)** Diagrama de tiempos.

ser un componente físico colocado en la ruta de retroalimentación para introducir un retardo Δt entre los cambios en la salida Y y la señal de retroalimentación y. Sin embargo, en muchos circuitos la salida Y sólo alimenta en forma directa a la entrada y, en cuyo caso Δt representa la suma de los retardos a través de las compuertas lógicas combinatorias, que determina el tiempo entre los cambios en la entrada x o la señal de retroalimentación y y los cambios correspondientes en la salida Y.

Este tipo de circuito secuencial asíncrono de nivel es tal vez el más difícil de analizar. La figura 10.18b muestra un diagrama de tiempos para el circuito. Las ecuaciones lógicas para la red son

$$Y = \overline{xy} = \bar{x} + \bar{y}$$
$$z = xy + \bar{x}\bar{y} = x \odot y$$

Al examinar la ecuación para Y, determinamos que el complemento de la señal de entrada x sirve como variable de control para las transiciones en el ciclo de retroalimentación de la línea de retardo. Si $x = 0$ ($\bar{x} = 1$), entonces $Y = 1$; es decir, el estado siguiente Y es independiente del estado actual y. Cuando $x = 1$ ($\bar{x} = 0$), el estado siguiente Y es el complemento del estado actual y; es decir, $Y = \bar{y}$, y el circuito secuencial pasa del estado 0 al 1 y viceversa.

Este tipo de circuitos es único entre los demás tipos de circuitos analizados con anterioridad, ya que no existen pulsos que nos ayuden en el análisis. Las transiciones de estado en los circuitos secuenciales síncronos son disparadas por pulsos dc reloj y, en los circuitos asíncronos de modo pulso, por pulsos sobre las entradas del circuito. Obtenemos con facilidad una tabla y un diagrama de estados para estos circuitos examinando el estado del circuito y las entradas en el instante preciso de cada transición activa de la señal de reloj o del pulso de entrada. Sin embargo, los circuitos asíncronos de nivel son más difíciles de describir. Por conveniencia, introduciremos una notación especial para manejar este caso poco usual.

Podemos describir el modelo genérico del circuito de modo fundamental de la figura 10.2 mediante el siguiente conjunto de ecuaciones lógicas en el instante t:

$$z_i^t = g_i(x_1^t, \ldots, x_n^t, y_1^t, \ldots, y_r^t), \qquad i = 1, \ldots, m \tag{10.1}$$

$$Y_j^t = h_i(x_1^t, \ldots, x_n^t, y_1^t, \ldots, y_r^t), \qquad j = 1, \ldots, r \tag{10.2}$$

$$y_j^{t+\Delta t} = Y_j^t, \qquad j = 1, \ldots, r \tag{10.3}$$

donde

$\mathbf{x} = (x_1,\ldots,x_n)$ = estado de entrada
$\mathbf{y} = (y_1,\ldots,y_n)$ = estado secundario
$\mathbf{z} = (z_1,\ldots,z_n)$ = estado de salida
$\mathbf{Y} = (Y_1,\ldots,Y_n)$ = estado de excitación
$(\mathbf{x},\mathbf{y})$ = estado total

También podemos escribir las ecuaciones como

$$\mathbf{z}^t = \mathbf{g}(\mathbf{x}^t, \mathbf{y}^t) \tag{10.4}$$

$$\mathbf{Y}^t = \mathbf{h}(\mathbf{x}^t, \mathbf{y}^t) \tag{10.5}$$

$$\mathbf{y}^{t+\Delta t} = \mathbf{Y}^t \tag{10.6}$$

10.4.1 Introducción

Como introducción al procedimiento de análisis de circuitos de modo fundamental, examinemos el circuito de la figura 10.19a. Este circuito queda descrito por el siguiente conjunto de ecuaciones:

$$z^t = g(x_1^t, x_2^t, y^t) = x_1^t x_2^t + \bar{x}_2^t y^t$$

$$Y^t = z^t$$

$$y^{t+\Delta t} = Y^t$$

donde

(x_1,x_2) = estado de entrada
(y) = estado secundario
(x_1,x_2,y) = estado total
(z) = estado de salida
(Y) = estado de excitación

La figura 10.19b muestra el diagrama de tiempos del circuito para una secuencia de entrada típica. En el capítulo 8 presentamos la construcción de diagramas de tiempos y los procedimientos ahí analizados siguen siendo válidos.

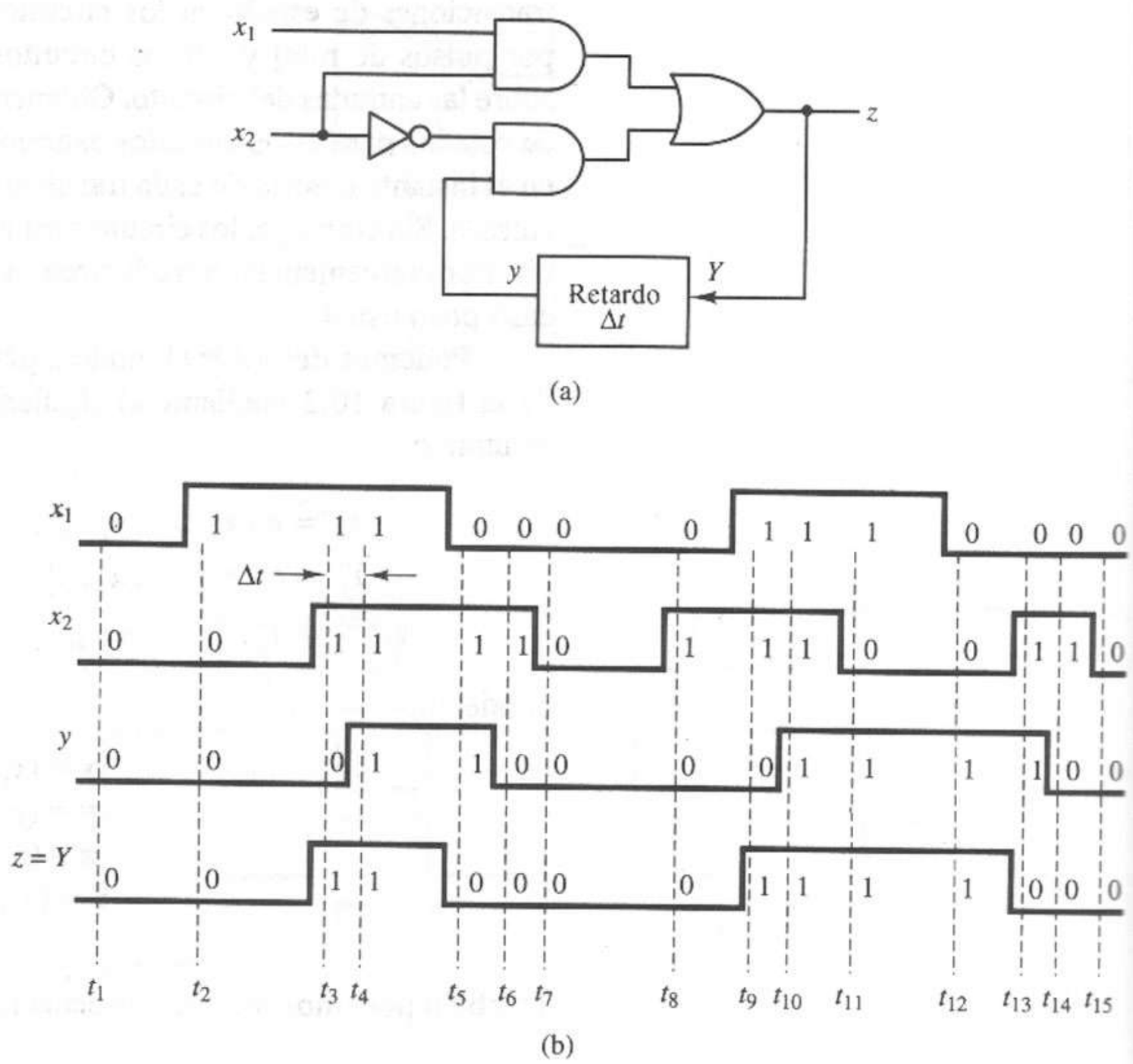


Figura 10.19 Circuito de modo fundamental. **(a)** Diagrama de circuito. **(b)** Diagrama de tiempos.

En la figura 10.19b podemos ver una situación de particular interés en el instante t_3. Observe que Y cambia de 0 a 1 en respuesta al cambio de 0 a 1 en x_2. Sin embargo, y no sigue con un cambio de 0 a 1 hasta el instante t_4. Este retardo en la respuesta de y se debe al elemento de retardo incluido en la ruta de retroalimentación. Como hemos supuesto un retardo de Δt, $t_4 - t_3 = \Delta t$.

Decimos que existe un *estado inestable* en el instante t_3 cuando $y \neq Y$. Existen otros estados inestables en los instantes t_5, t_9 y t_{13}. Cuando $y = Y$, tenemos un *estado estable*.

Debemos observar que los estados inestables aparecen durante un periodo de tiempo igual a Δt y, por tanto, tienen una naturaleza transitoria. Sin embargo, el comportamiento transitorio de los circuitos de modo fundamental es, en general, más crítico para el funcionamiento adecuado del dispositivo, así que lo estudiaremos con más detalle en una sección posterior del capítulo.

En resumen, un circuito de modo fundamental tiene un *estado estable* cuando se satisface la siguiente relación:

$$\mathbf{y}^t = \mathbf{Y}^t \tag{10.7}$$

Definimos un *estado inestable* mediante la relación

$$\mathbf{y}^t \neq \mathbf{Y}^t \tag{10.8}$$

y \ x_1x_2	00	01	11	10
0	⓪/0	⓪/0	1/1	⓪/0
1	①/1	0/0	①/1	①/0

Y/z

Figura 10.20 Tabla de excitación.

10.4.2 Representaciones tabulares

En muchos casos es conveniente representar los circuitos de modo fundamental en forma tabular. La primera forma a examinar es la *tabla de excitación*. Una tabla de excitación presenta el estado de excitación y el de salida en función del estado total $(x_1,...,x_n,y_1,...,y_r)$. Por tanto, la tabla de excitación es una representación tabular de las ecuaciones 10.1 y 10.2. La figura 10.20 muestra la tabla de excitación correspondiente a la figura 10.19a. Podemos construir esta tabla mediante una combinación adecuada de los mapas K para Y y z.

Observe que cada columna de la tabla está asociada con un único estado de entrada, y cada fila de la tabla corresponde a un único estado secundario. Por tanto, cada celda de la tabla representa un estado total único (x_1,x_2,y) del circuito. Cada celda contiene los estados de excitación y de salida dados para el estado total correspondiente por las ecuaciones 10.1 y 10.2 respectivamente. Indicamos los estados estables encerrando en un círculo el estado de excitación correspondiente.

En muchos casos es recomendable separar las funciones de excitación y de salida en dos tablas. Hemos hecho esto en la figura 10.21 para el circuito de la figura 10.19. Así, podemos representar la tabla de excitación con formato de una o dos tablas, y generalizar ambos formatos agregando las filas y columnas necesarias para acomodar los estados y entradas adicionales.

y \ x_1x_2	00	01	11	10	00	01	11	10
0	⓪	⓪	1	⓪	0	0	1	0
1	①	0	①	①	1	0	1	1
	Y				z			

Figura 10.21 Tabla de excitación (forma alternativa).

	x_1x_2 00	01	11	10
a	ⓐ/0	ⓐ/0	b/1	ⓐ/0
b	ⓑ/1	a/0	ⓑ/1	ⓑ/1

Figura 10.22 Tabla de flujo.

La *tabla de flujo* es otra representación útil de un circuito de modo fundamental. Las tablas de flujo son similares a las tablas de excitación; sin embargo, en una tabla de flujo los estados de excitación y los estados secundarios están representados por letras u otros caracteres no binarios. Por tanto, una tabla de flujo especifica el comportamiento del circuito, pero no especifica la realización del circuito. La figura 10.22 presenta una tabla de flujo para el circuito de la figura 10.19a.

Podemos utilizar las tablas de flujo o las tablas de excitación para determinar la respuesta de salida de un circuito a una secuencia de entrada dada. Sin embargo, la tabla de excitación proporciona el comportamiento del estado secundario y de excitación como información adicional.

La tabla de flujo de la figura 10.23 ilustra el flujo correspondiente al intervalo de tiempo de t_1 a t_7 en el diagrama de tiempos de la figura 10.19b. Observe la aparición de estados inestables en la secuencia de flujo. Además, observe que un cambio en la entrada provoca un movimiento horizontal en la tabla de flujo. Los movimientos verticales se producen por cambios en los estados secundarios que son consecuencia de los cambios en la entrada. En otras palabras, un cambio en la entrada hace que el circuito se desplace horizontalmente a una

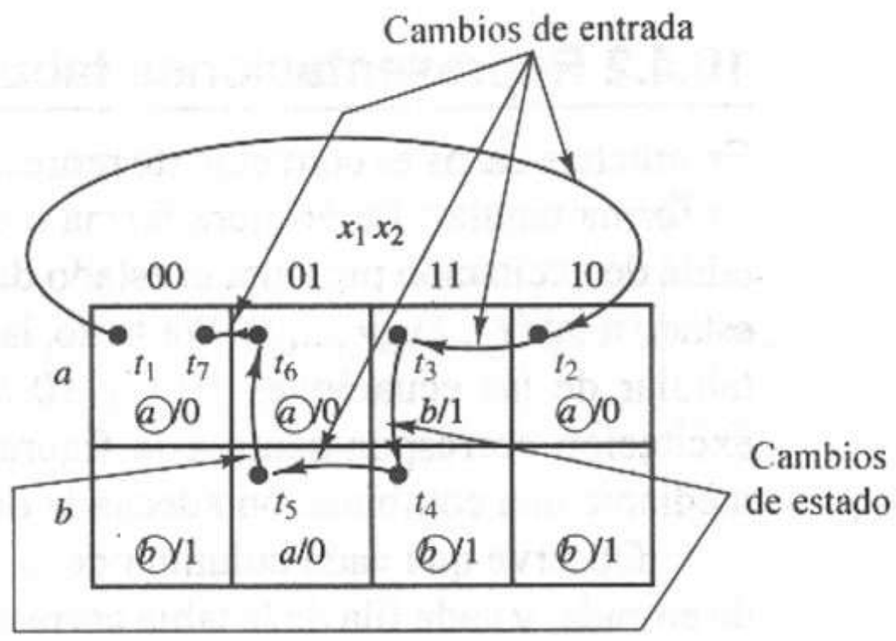


Figura 10.23 Tabla de flujo que muestra la secuencia de flujo.

nueva columna y después verticalmente dentro de la columna hasta encontrar un estado estable.

10.4.3 Procedimiento de análisis

El ejemplo anterior sugiere el siguiente procedimiento de análisis.

1. Determine las ecuaciones de excitación y de salida a partir del diagrama de circuitos.
2. Grafique los mapas K de excitación y de salida para Y y z, y construya la tabla de excitación a partir de estos mapas K.
3. Localice y encierre en un círculo todos los estados estables de la tabla de excitación.
4. Asigne un símbolo no binario único a cada fila de la tabla de excitación. Con frecuencia se utilizan letras o los equivalentes decimales de los códigos de los estados secundarios.
5. Construya la tabla de flujo, remplazando cada estado binario en la tabla de excitación por un símbolo que represente su asignación de estado.

Ilustraremos este procedimiento a continuación mediante el siguiente ejemplo.

EJEMPLO 10.6

Deducir la tabla de flujo para el circuito de la figura 10.24a.

Paso 1. Las ecuaciones de excitación y de salida para la red son

$$Y_1 = \bar{x}\bar{y}_2$$
$$Y_2 = x\bar{y}_1$$
$$z = \bar{x}y_1$$

Paso 2. Construimos una tabla de excitación a partir de los mapas K de Y_1, Y_2 y z, como se muestra en la figura 10.25a. La tabla resultante aparece en la figura 10.25b.

Paso 3. Podemos localizar los estados estables mediante la condición $y_1y_2 = Y_1Y_2$. Encerramos en un círculo estos estados, como se muestra en la figura 10.25b.

Paso 4. Elegimos los equivalentes decimales en los códigos de estado secundario, más 1, para representar las filas correspondientes de la tabla de excitación; por ejemplo, $y_1y_2 = (01) = 2$ en la figura 10.25b.

Paso 5. La figura 10.25c muestra la tabla de flujo resultante. Observe que hemos intercambiado las dos últimas filas para mantener el orden numérico.

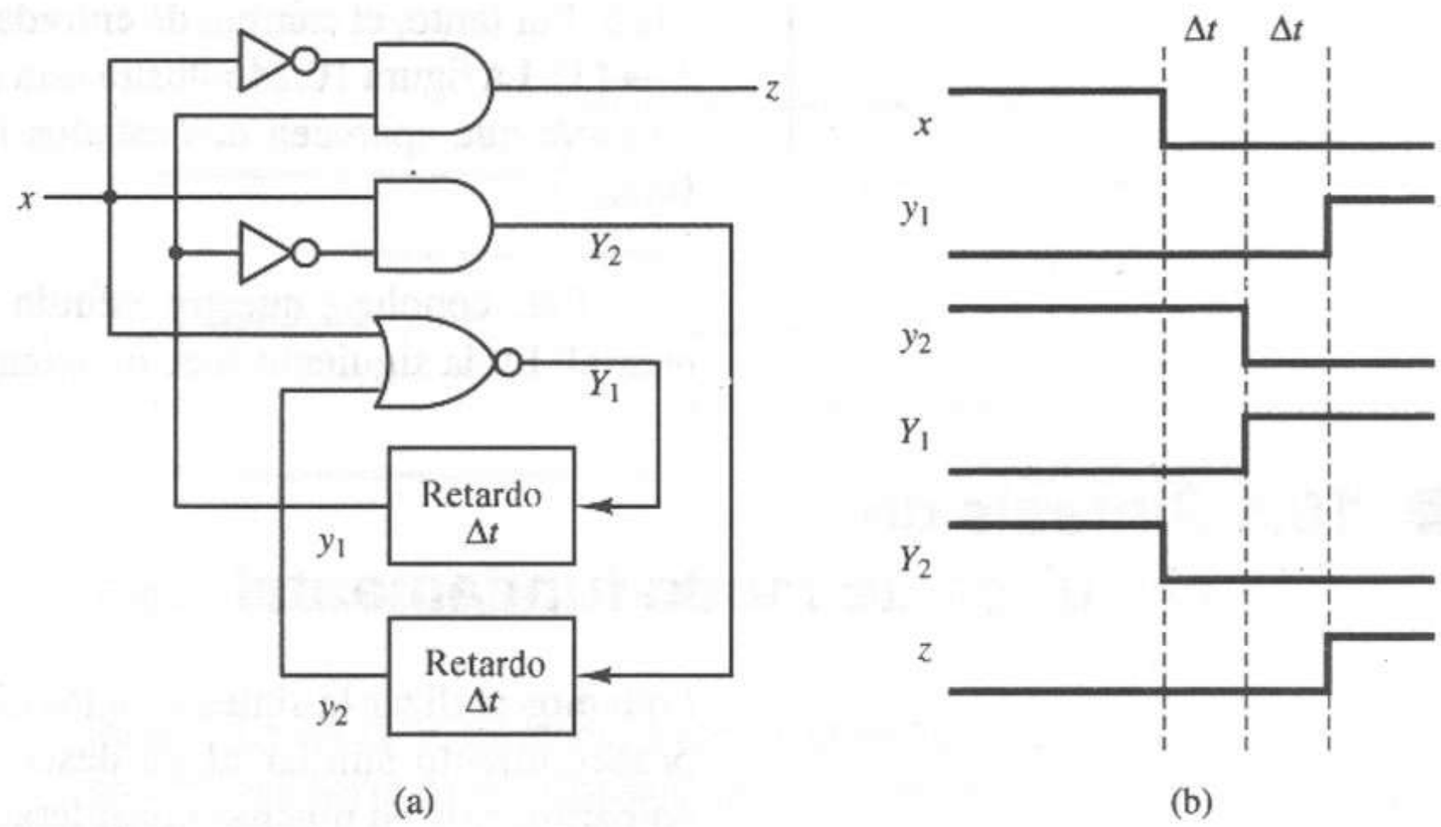


Figura 10.24 Ejemplo de análisis de modo fundamental. **(a)** Diagrama de circuito. **(b)** Diagrama de tiempos.

y_1y_2 \ x	0	1
00	**1**	**0**
01	**0**	**0**
11	**0**	**0**
10	**1**	**0**

Y_1

y_1y_2 \ x	0	1
00	**0**	**1**
01	**0**	**1**
11	**0**	**0**
10	**0**	**0**

Y_2

y_1y_2 \ x	0	1
00	**0**	**0**
01	**0**	**0**
11	**1**	**0**
10	**1**	**0**

z

(a)

y_1y_2 \ x	0	1
1 = 00	10/0	01/0
2 = 01	00/0	(01)/0
4 = 11	00/1	00/0
3 = 10	(10)/1	00/0

Y_1Y_2/z

(b)

x	0	1
1	3/0	2/0
2	1/0	(2)/0
3	(3)/1	1/0
4	1/1	1/0

(c)

Figura 10.25 Tablas de excitación y de flujo deducidas de los mapas K. **(a)** Mapas K. **(b)** Tabla de excitación. **(c)** Tabla de flujo.

Es interesante observar los cambios de estado que se presentan en el circuito cuando éste se desplaza con respecto del estado estable (2) cuando la entrada cambia de 1 a 0. (*Nota*: En el análisis siguiente, utilizaremos los paréntesis para denotar los estados estables.) Primero, el circuito pasa del estado estable (2) al estado inestable 1 de la fila 2. Después, el circuito se desplaza al estado inestable 3 de la fila 1. Por último, el dispositivo es transferido al estado estable (3) de la

fila 3. Por tanto, el cambio de entrada inicia una secuencia de estados (2) → 1 → 3 → (3). La figura 10.24b ilustra esta secuencia en forma de diagrama de tiempos. Observe que aparecen dos estados inestables antes de llegar al estado estable final.

Esto concluye nuestro estudio del análisis de un circuito de modo fundamental. En la siguiente sección examinaremos la síntesis de tales circuitos.

10.5 Síntesis de circuitos de modo fundamental

Podemos realizar la síntesis de los circuitos de modo fundamental mediante un procedimiento similar al ya descrito para los circuitos de modo pulso. Sin embargo, existen muchas consideraciones de diseño particulares en el caso del modo fundamental y les daremos un tratamiento especial más adelante.

Los circuitos de modo fundamental no se pueden representar de manera conveniente mediante diagramas o tablas de estados, ya que el estado total queda determinado por el estado de entrada y el estado secundario. La tabla de flujo primitiva es una representación alternativa de la tabla y el diagrama de estados, que se aplica a los dispositivos de modo fundamental. Una *tabla de flujo primitiva* es una tabla de flujo que sólo contiene un estado estable por fila.

Ahora definiremos un procedimiento de síntesis para los circuitos de modo fundamental. Ilustraremos este procedimiento mediante dos ejemplos. En una sección posterior de este capítulo veremos un tercer ejemplo.

10.5.1 Procedimiento de síntesis

Paso 1. Construir una tabla de flujo primitiva a partir de una descripción verbal del circuito por realizar.

Paso 2. Determinar una tabla de flujo reducida a partir de la tabla de flujo primitiva.

Paso 3. Realizar una asignación de estados secundarios.

Paso 4. Construir la tabla de excitación y de salida. Más adelante estableceremos reglas especiales para la construcción de la tabla de salida.

Paso 5. Determinar las ecuaciones lógicas para cada variable de estado de excitación y de salida.

Paso 6. Realizar las ecuaciones lógicas con los dispositivos lógicos adecuados.

EJEMPLO 10.7

Diseñar un circuito secuencial asíncrono de dos entradas (x_1,x_2) y una salida (z) que cumpla lo siguiente. Siempre que $x_1 = 0$, $z = 0$. El primer cambio en la entrada x_2 que se presente mientras $x_1 = 1$ debe hacer que la salida sea $z = 1$. Una salida $z = 1$ no debe cambiar a $z = 0$ hasta que $x_1 = 0$. La figura 10.26 muestra una respuesta entrada/salida típica del circuito dado.

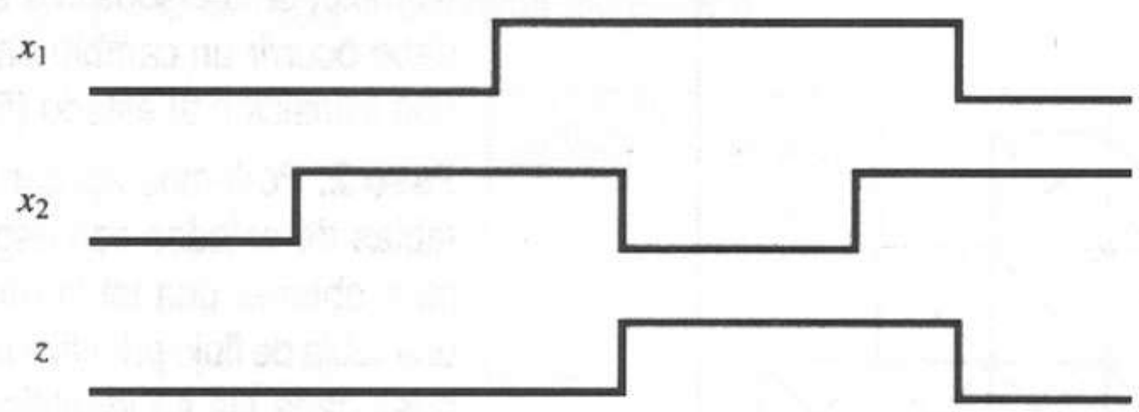


Figura 10.26 Respuesta entrada/salida típica.

	x_1x_2 00	01	11	10
1	(1)/0	2/0	–/–	6/0
2	1/0	(2)/0	3/0	–/–
3	–/–	2/0	(3)/0	4/–
4	1/–	–/–	5/1	(4)/1
5	–/–	2/–	(5)/1	4/1
6	1/0	–/–	5/–	(6)/0

Figura 10.27 Tabla de flujo primitiva.

Paso 1. La figura 10.27 muestra una tabla de flujo primitiva que satisface las condiciones del circuito. Debemos observar varias características de la tabla. Primero, existe una única columna para cada combinación de entrada. Además, cada fila contiene un estado estable con una salida específica, dos estados inestables con salidas no especificadas y una columna con un estado y una salida no especificados. Ésta última es siempre la columna en la que los valores de las dos entradas difieren de los valores de la columna que contiene el estado estable. Como el funcionamiento en modo fundamental sólo permite que se presente un cambio de entrada en un instante dado, no pueden ocurrir transiciones correspondientes a dos o más cambios de entrada. Por tanto, no se especifican los estados siguientes. Especificamos las salidas de los estados inestables como sigue:

1. Asignamos una salida 0 a cada estado inestable que sea un estado transitorio entre dos estados estables, cada uno de los cuales tiene asociada una salida 0.
2. Asignamos una salida 1 a cada estado inestable que sea un estado transitorio entre dos estados estables, cada uno de los cuales tiene asociada una salida 1.
3. Asignamos una condición prescindible a cada estado inestable que sea un estado transitorio entre dos estados estables, uno de los cuales tiene una salida 0 y el otro una salida 1.

Al asignar las salidas de esta manera evitamos los cambios momentáneos en la salida cuando el circuito pasa por los estados inestables.

Ahora explicaremos la necesidad de cada uno de los estados especificados. Suponga que el dispositivo se encuentra en el estado (1) y que x_2 cambia de 0 a 1. No debe ocurrir ningún cambio en la salida en estas condiciones. Por tanto, se llega al estado (2) por vía del estado inestable 2. Un cambio en x_1 de 0 a 1, mientras el dispositivo se encuentra en el estado (2), tampoco produce un cambio en la salida. Por tanto, se ingresa en el estado (3) a través del estado inestable 3. Si el circuito se encuentra en el estado (3) y x_2 cambia de 1 a 0, debe haber un cambio en la salida de 0 a 1. Este cambio se produce al crear el estado (4) con la salida $z = 1$. Ahora, supongamos que el dispositivo se encuentra en el estado (4) y que x_2 cambia de 0 a 1. En esta situación, no debe ocurrir ningún cambio en la salida, ya que $x1 \neq 0$. Por tanto, en la columna 11 debe haber un estado estable con $z = 1$. El estado (5) satisface esta condición. Necesitamos el estado (6), ya que es necesario un estado estable con $z = 0$ en la columna 10 cuando haya un cambio en la entrada de 00 a 10.

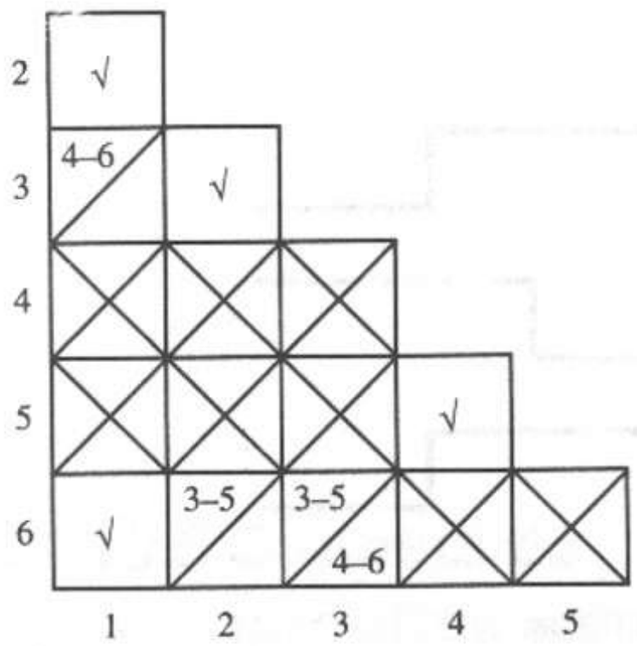


Figura 10.28 Tabla de implicación.

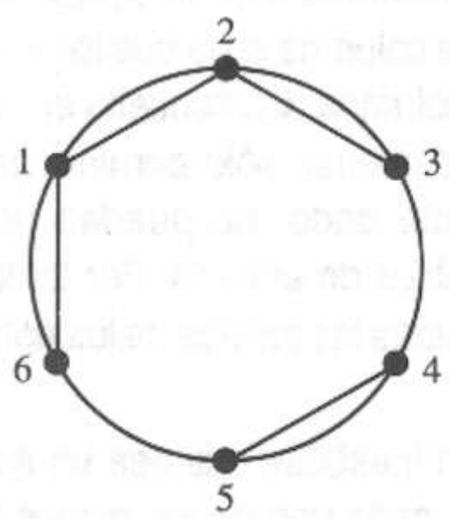


Figura 10.29 Diagrama de fusión.

Podemos acomodar las transiciones restantes sin definir más estados. Por ejemplo, si el dispositivo se encuentra en el estado (6) y x_2 cambia de 0 a 1, debe ocurrir un cambio en la salida de 0 a 1. Podemos realizar esto mediante una transición al estado (5).

Paso 2. Podemos aplicar los métodos desarrollados para la reducción de las tablas de estados con especificación incompleta a una tabla de flujo primitiva para obtener una tabla de flujo reducida. La ausencia de estados actuales en una tabla de flujo primitiva no representa un problema en el proceso de reducción, pues cada fila se identifica con un estado estable único. Así, el concepto de estados compatibles es remplazado por el concepto de filas compatibles. Dos filas son compatibles (se pueden fusionar) si sus estados y salidas son compatibles en cada columna de la tabla de flujo primitiva. Determinamos la compatibilidad de los estados estables e inestables como sigue. El estado estable (i) y el estado inestable i son compatibles. El estado estable (i) y el estado inestable j son compatibles si (i) es compatible con (j). El estado inestable i es compatible con el estado inestable j si (i) es compatible con (j).

La figura 10.28 muestra la tabla de implicación correspondiente a la tabla de flujo primitiva de la figura 10.27. Los pares de filas compatibles son (1,2), (1,6), (2,3) y (4,5). Podemos construir un diagrama de fusión como se muestra en la figura 10.29 para ilustrar las fusiones posibles.

Concluimos la reducción de la tabla de flujo al seleccionar una cubierta cerrada mínima. Para este ejemplo, los requisitos de implicación son triviales. Por tanto, el problema se reduce a la elección de una cubierta mínima. Una cubierta mínima obvia es {(1,6),(2,3),(4,5)}, la cual conduce a la tabla de flujo reducida de la figura 10.30.

Podemos obtener la tabla de flujo equivalente de la figura 10.31 reetiquetando las filas como a, b y c.

	x_1x_2 00	01	11	10
(1, 6)	①/0	2/0	5/–	⑥/0
(2, 3)	1/0	②/0	③/0	4/–
(4, 5)	1/–	2/–	⑤/1	④/1

Figura 10.30 Tabla de flujo reducida.

	x_1x_2 00	01	11	10
a	ⓐ/0	b/0	c/–	ⓐ/0
b	a/0	ⓑ/0	ⓑ/0	c/–
c	a/–	b/–	ⓒ/1	ⓒ/1

Figura 10.31 Tabla de flujo reducida con estados reetiquetados.

Paso 3. Debemos asignar a cada fila de la tabla de flujo reducida un único código del estado secundario. (Esta asignación debe cumplir ciertos criterios que analizaremos en la siguiente sección.) Por ahora, consideraremos aceptable

una elección arbitraria. En este ejemplo, necesitamos dos variables de estado (y_1,y_2). Utilizamos la siguiente asignación:

Fila	y_1 y_2
a	00
b	11
c	01

Paso 4. Construimos la tabla de excitación a partir de la tabla de flujo reducida, remplazando cada letra por el código de estado secundario asignado en el paso anterior. Encerramos en un círculo los estados estables. Para este ejemplo, la figura 10.32a muestra la tabla de excitación.

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	00	11	01	00
11	00	11	11	01
01	00	11	01	01

$Y_1 Y_2$

(a)

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	0	0	d	0
11	0	0	0	d
01	d	d	1	1

z

(b)

Figura 10.32 Tablas de excitación y de salida. **(a)** Tabla de excitación. **(b)** Tabla de salida.

Obtenemos la tabla de salida de la figura 10.32b de la tabla de flujo reducida.

Paso 5. Obtenemos las ecuaciones lógicas para cada variable de estado de excitación y de salida transfiriendo la información de las tablas de excitación y de salida a mapas K y deduciendo después las ecuaciones lógicas. La figura 10.33 muestra los mapas para el ejemplo, de donde obtenemos las siguientes ecuaciones de excitación y de salida:

$$Y_1 = \bar{x}_1 x_2 + x_2 y_1$$
$$Y_2 = x_2 + x_1 y_2$$
$$z = \bar{y}_1 y_2$$

Paso 6. La figura 10.34 muestra una realización del circuito con compuertas AND, OR y NOT.

EJEMPLO 10.8

Diseñar un circuito de modo fundamental con dos entradas (x_1,x_2) y dos salidas (z_1,z_2) con las siguientes especificaciones. Cuando $x_1x_2 = 00$, $z_1z_2 = 00$. Obtenemos la salida 10 cuando se

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	1	0
10	d	d	d	d

Y_1

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	1
11	0	1	1	1
10	d	d	d	d

Y_2

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	0	0	d	0
01	d	d	1	1
11	0	0	0	d
10	d	d	d	d

z

Figura 10.33 Mapas K para las tablas de excitación y de salida.

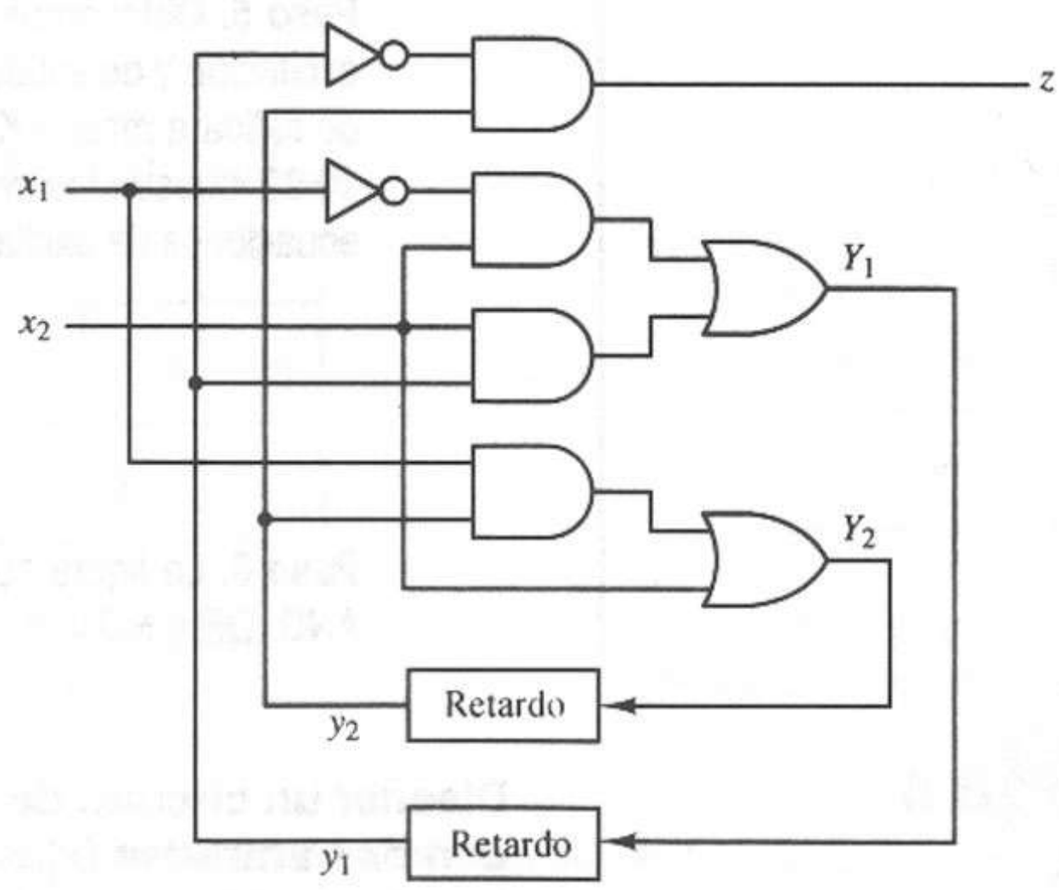


Figura 10.34 Realización del ejemplo de síntesis.

presente la secuencia de entrada 00-01-11. La salida se mantiene en 10 hasta que la entrada regresa a 00, en cuyo caso asume el valor 00. Obtenemos una salida 01 al recibirse la secuencia de entrada 00-10-11. De nuevo, la salida se mantiene en 01 hasta que ocurre una entrada 00, que regresa la salida a 00.

Paso 1. Al construir la tabla de flujo primitiva, es útil observar que debemos definir al menos un estado estable en cada columna para cada salida posible que podamos obtener mediante las entradas respectivas. Esta observación produce la tabla de flujo primitiva parcialmente completa de la figura 10.35a. Podemos completar la tabla de flujo estableciendo las transiciones necesarias entre los estados estables. Podemos llevar a cabo tales transiciones especificando los estados inestables, como lo muestra la tabla de flujo primitiva completa de la figura 10.35b.

	x_1x_2 00	01	11	10
1	1/00		–/–	
2		2/00		–/–
3		3/10		–/–
4		4/01		–/–
5	–/–		5/10	
6	–/–		6/01	
7		–/–		7/00
8		–/–		8/10
9		–/–		9/01

(a)

	x_1x_2 00	01	11	10
1	1/00	2/00	–/*dd*	7/00
2	1/00	2/00	5/*d*0	–/*dd*
3	1/*d*0	3/10	5/10	–/*dd*
4	1/0*d*	4/01	6/01	–/*dd*
5	–/*dd*	3/10	5/10	8/10
6	–/*dd*	4/01	6/01	9/01
7	1/00	–/*dd*	6/0*d*	7/00
8	1/*d*0	–/*dd*	5/10	8/10
9	1/0*d*	–/*dd*	6/01	9/01

(b)

Figura 10.35 Preparación de la tabla de flujo primitiva. **(a)** Tabla de flujo primitiva parcial. **(b)** Tabla de flujo primitiva terminada.

Paso 2. Iniciamos la reducción de la tabla de flujo construyendo la tabla de implicación como en la figura 10.36. De la tabla de implicación, vemos que los pares de filas compatibles son (1,2), (1,7), (2,8), (3,5), (3,8), (4,6), (4,7), (4,9),

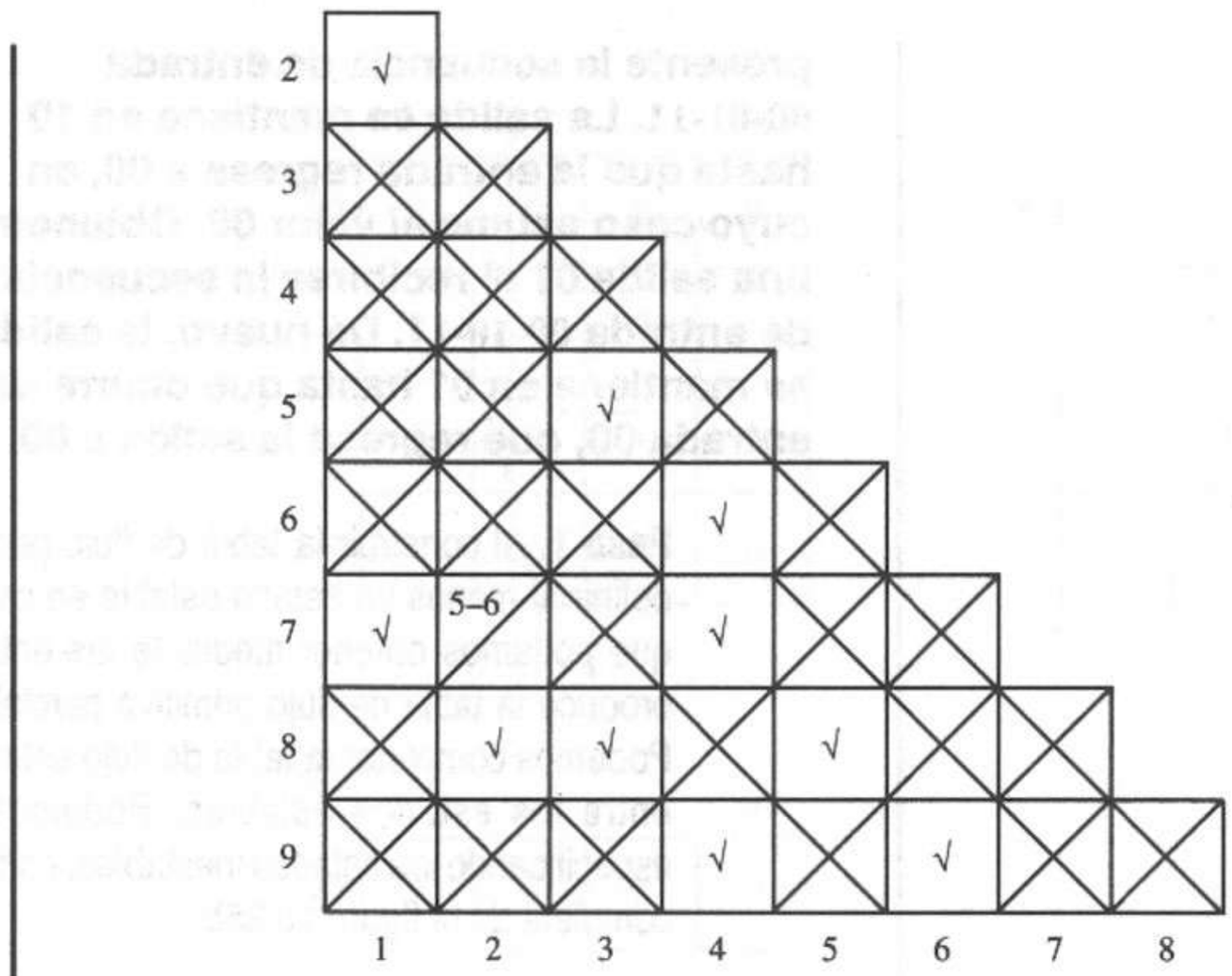


Figura 10.36 Tabla de implicación.

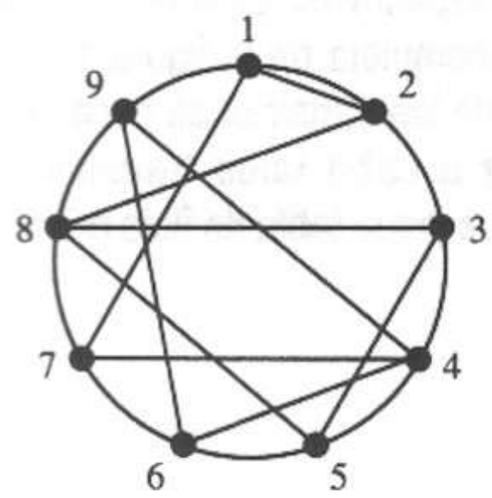


Figura 10.37 Diagrama de fusión.

(5,8) y (6,9). El diagrama de fusión de la figura 10.37 es muy útil para elegir una cubierta.

Al elegir una cubierta cerrada mínima obtenemos el siguiente conjunto:

$$\{(1,2),(1,7),(3,5,8),(4,6,9)\}$$

Utilizamos la cubierta para obtener la tabla de flujo reducida de la figura 10.38, donde *a* es (1,2), *b* es (3,5,8), *c* es (1,7) y *d* es (4,6,9). Observe que la fila 1 se incluye en dos estados, *a* y *c*. Por tanto, podemos reemplazar cualquier estado inestable de la tabla de flujo primitiva cuyo estado siguiente sea 1 por un estado siguiente de *a* o de *c* en la tabla de flujo reducida, de manera arbitraria.

Paso 3. Utilizaremos la siguiente asignación de estados:

Row	y_1y_2
a	00
b	01
c	10
d	11

	x_1x_2 00	01	11	10
a	(*a*)/00	(*a*)/00	*b*/*d*0	*c*/00
b	*a*/*d*0	(*b*)/10	(*b*)/10	(*b*)/10
c	(*c*)/00	*a*/00	*d*/0*d*	(*c*)/00
d	*c*/0*d*	(*d*)/01	(*d*)/01	(*d*)/01

Figura 10.38 Tabla de flujo reducida.

Paso 4. La figura 10.39 muestra las tablas de excitación y de salida.

Paso 5. Podemos obtener las siguientes ecuaciones de excitación y de salida mediante los mapas K de la figura 10.40.

$$Y_1 = y_1y_2 + x_1y_1 + \bar{x}_2y_1 + x_1\bar{x}_2\bar{y}_2$$
$$Y_2 = x_1x_2 + x_1y_2 + x_2y_2$$
$$z_1 = \bar{y}_1y_2$$
$$z_2 = y_1y_2$$

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10	00	01	11	10
00	00	00	01	10	00	00	$d0$	00
01	00	01	01	01	$d0$	10	10	10
10	10	00	11	10	00	00	$0d$	00
11	10	11	11	11	$0d$	01	01	01
	$Y_1 Y_2$				$z_1 z_2$			

Figura 10.39 Tablas de excitación y de salida.

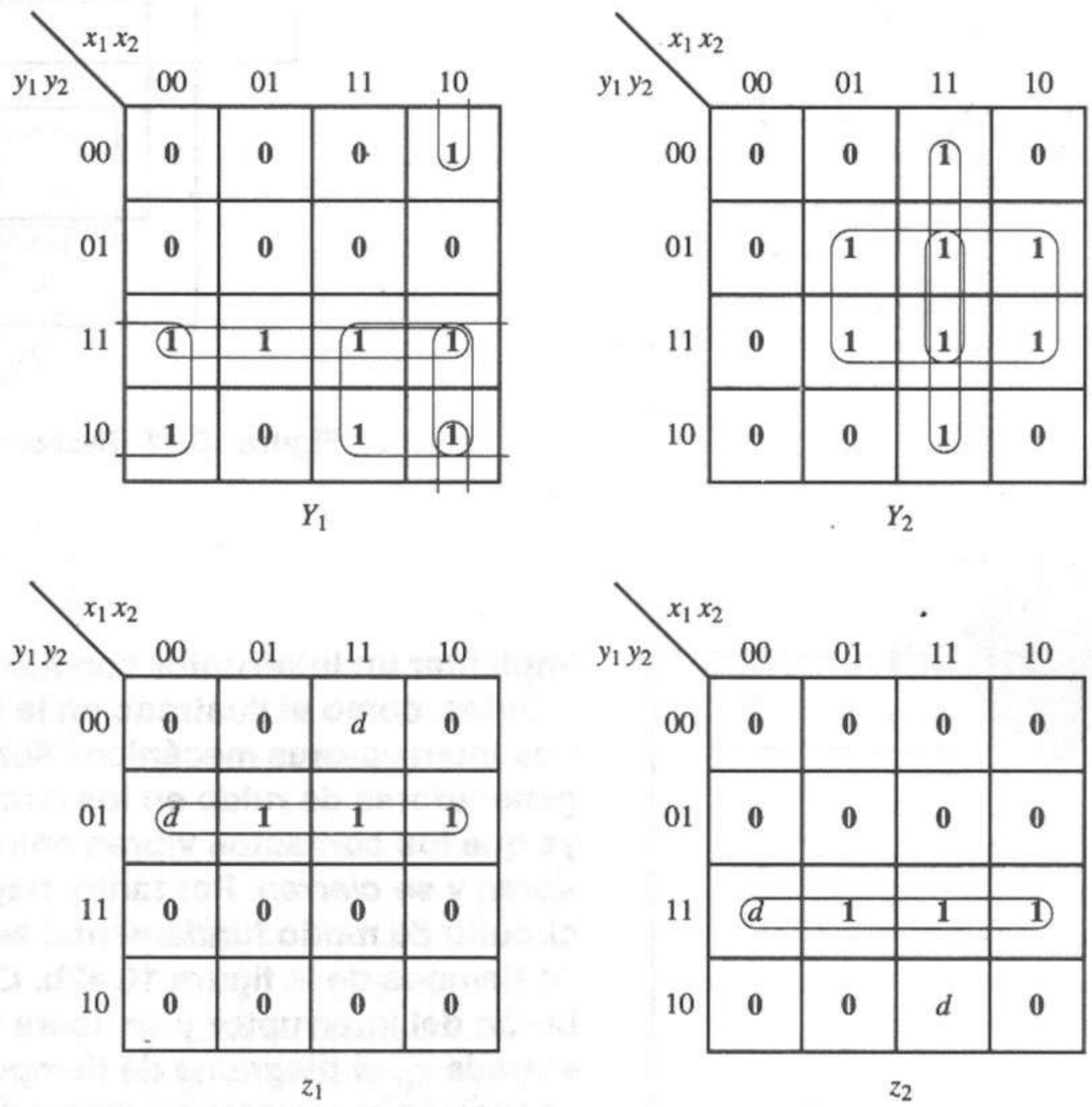


Figura 10.40 Mapas K para las tablas de excitación y de salida.

Paso 6. La figura 10.41 muestra la realización del circuito.

A veces, podemos implantar un diseño de circuito de modo fundamental sin deducir una tabla de flujo primitiva, como lo muestra el siguiente ejemplo.

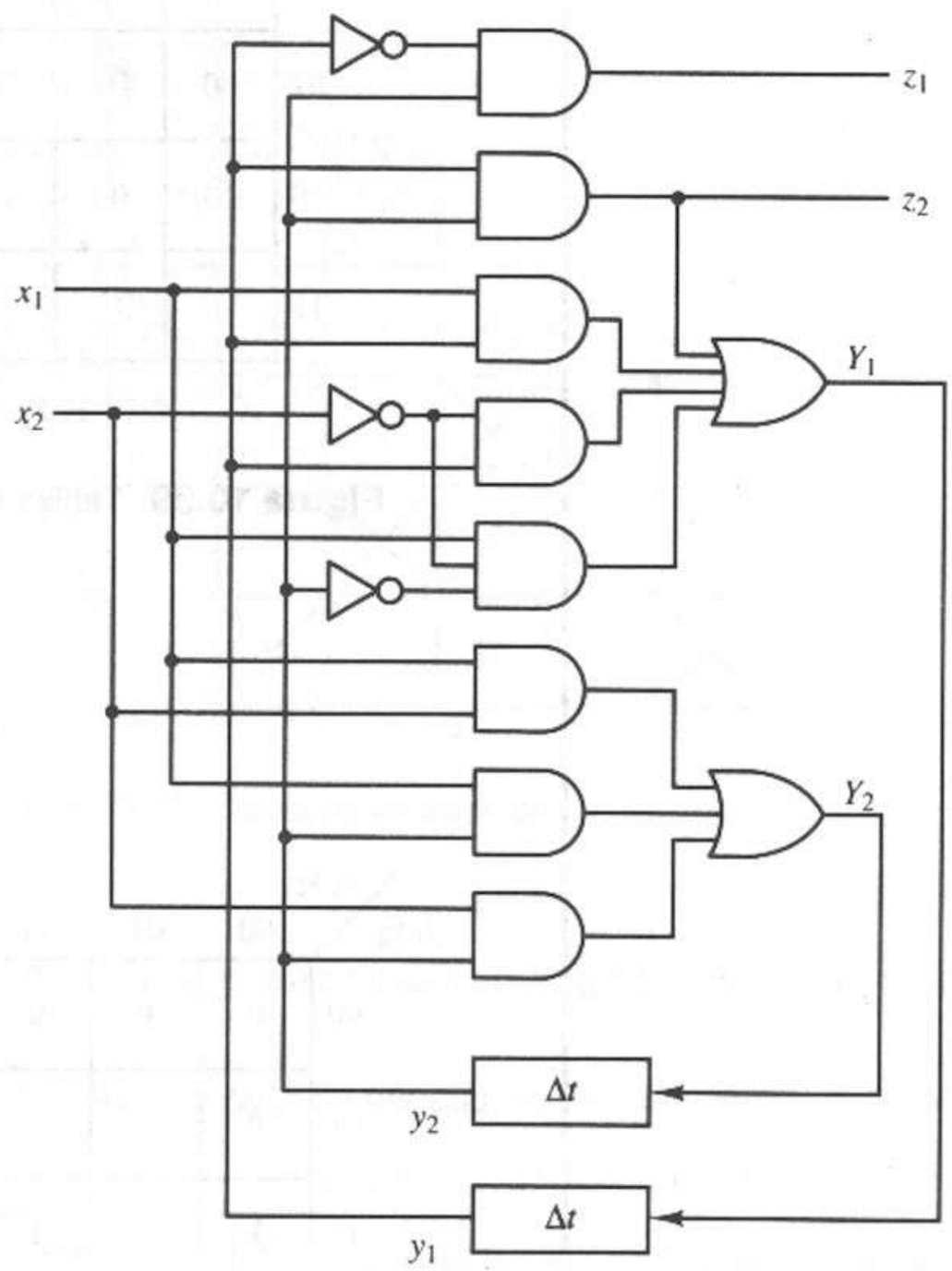


Figura 10.41 Realización del ejemplo de diseño.

EJEMPLO 10.9

Implantar un interruptor con filtrado de rebotes, como el ilustrado en la figura 10.42a. Los interruptores mecánicos son notables generadores de ruido en los circuitos digitales, ya que los contactos vibran entre sí cuando se *abren* y se *cierran*. Por tanto, hay que diseñar un circuito de modo fundamental según el diagrama de tiempos de la figura 10.42b. Cuando se pulsa el botón del interruptor y se libera el contacto en la entrada x_1, el diagrama de tiempos muestra una oscilación momentánea. Después de un periodo corto de tiempo (por lo general unos cuantos milisegundos en el caso de los interruptores comerciales) la señal x_1 se estabiliza en 1 lógico. Después, cuando se cierran los contactos en x_2, éstos oscilan inicialmente y por último x_2 asume el valor 0 lógico. Al liberarse el botón, se presenta la misma secuencia de eventos pero en orden inverso.

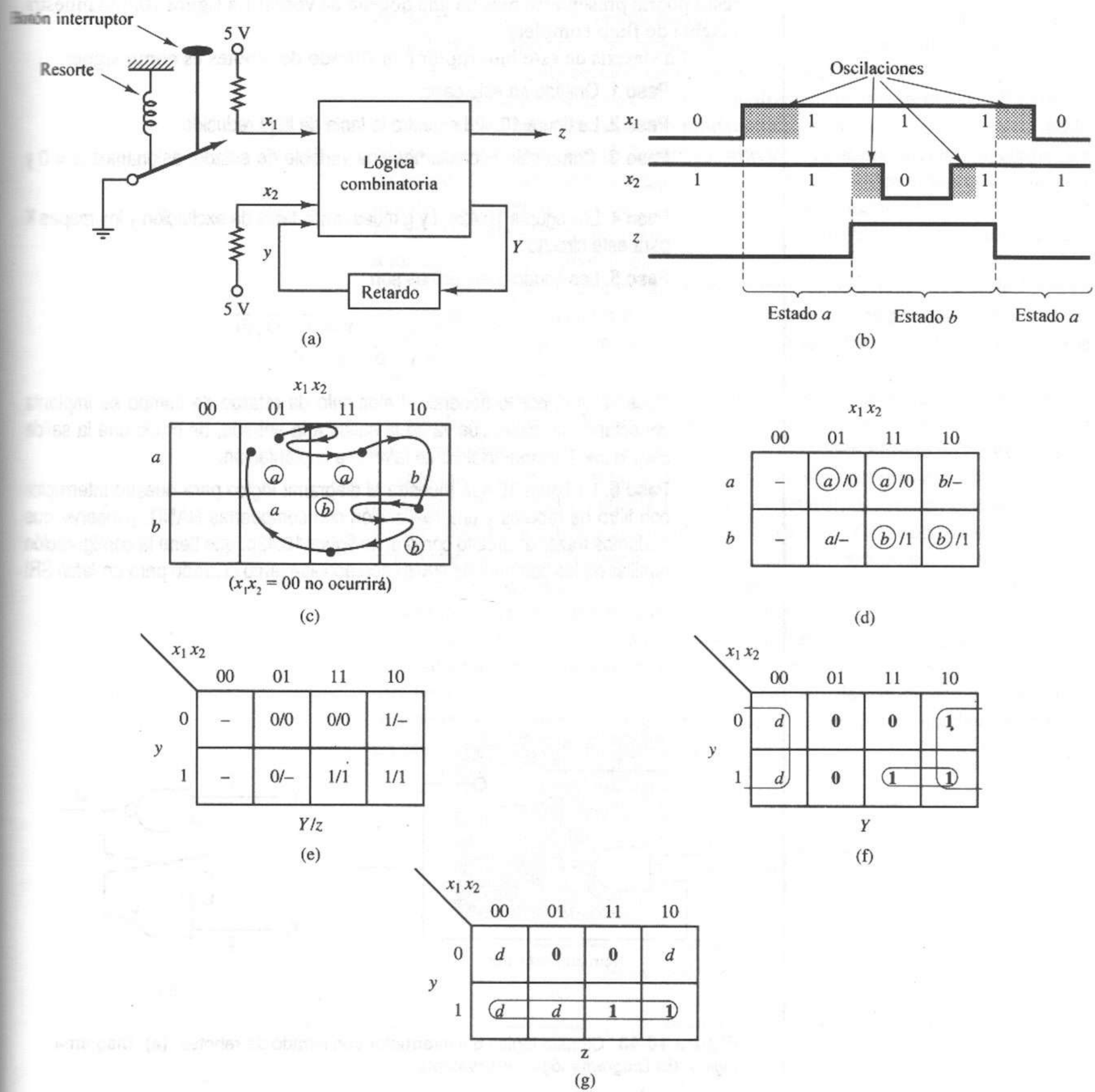


Figura 10.42 Diseño del interruptor con filtrado de rebotes. **(a)** Interruptor mecánico con filtrado de rebotes. **(b)** Comportamiento de tiempos requerido. **(c)** Secuencia de flujo requerida. **(d)** Tabla de flujo reducida completa. **(e)** Tabla de excitación. **(f)** Mapa K para *Y*. **(g)** Mapa K para *z*.

La figura 10.42c muestra el ejemplo de una tabla de flujo (sin presentar la salida z) que ilustra el comportamiento de estados buscado. Las oscilaciones en las entradas se "filtran" permitiendo que el circuito rebote hacia atrás y hacia adelante entre los estados estables ⓐ de la primera fila y ⓑ de la segunda. Las oscilaciones aparecen como un ciclo simple en la tabla de flujo, pero en un interruptor real, ¡esto podría presentarse más de una docena de veces! La figura 10.42d muestra la tabla de flujo completa.

La síntesis de este interruptor con filtrado de rebotes es como sigue:

Paso 1. Omitido en este caso.

Paso 2. La figura 10.42d muestra la tabla de flujo reducida.

Paso 3. Como sólo necesitamos una variable de estado, asignamos $a = 0$ y $b = 1$.

Paso 4. Las figuras 10.42e, f y g muestran la tabla de excitación y los mapas K para este circuito.

Paso 5. Las ecuaciones lógicas son

$$Y = \bar{x}_2 + x_1 y = \overline{x_2 \cdot (\overline{x_1 y})}$$
$$z = y \quad \text{o} \quad z = Y$$

Observe que, por lo general, el elemento de retardo de tiempo se implanta conectando un cable que va de la salida a la entrada, de modo que la salida $z = y$ o $z = Y$ es esencialmente la misma implantación.

Paso 6. La figura 10.43a muestra el diagrama lógico para nuestro interruptor con filtro de rebotes y una realización con compuertas NAND. ¡Observe que podemos trazar el circuito como en la figura 10.43b, que tiene la configuración familiar de las compuertas NAND con acoplamiento cruzado para un *latch* SR!

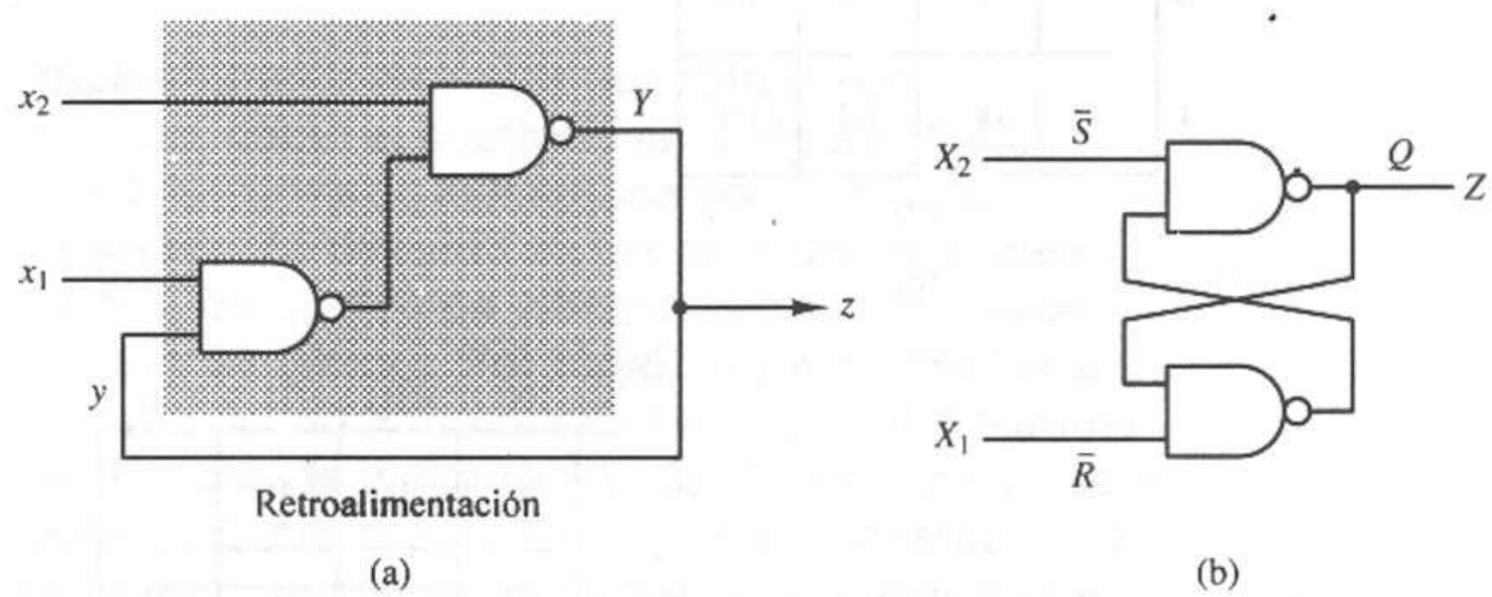


Figura 10.43 Circuito lógico del interruptor con filtrado de rebotes. **(a)** Diagrama lógico. **(b)** Diagrama lógico equivalente.

En la siguiente sección presentaremos un cuarto ejemplo de diseño que implica la elección de una asignación de estado secundario. Debemos tomar en cuenta los requisitos especiales al hacer esta asignación para un circuito de modo fundamental. Ahora analizaremos estos requisitos.

10.6 Introducción a las competencias, ciclos y riesgos

Las características de los componentes individuales con los que construimos un circuito lógico influyen en las características de desempeño del circuito. En particular, el tiempo de respuesta relativo de los componentes tiene un efecto significativo sobre el comportamiento de los circuitos asíncronos de modo fundamental. En esta sección estudiaremos tales efectos.

Sin embargo, antes de continuar, analizaremos brevemente las fuentes de retardos en los circuitos de modo fundamental. El modelo de circuito de modo fundamental de la figura 10.2 muestra rutas de retroalimentación con elementos de retardo. En todos los circuitos físicos tenemos retardos adicionales que surgen de los elementos lógicos y los cables de interconexión. Por tanto, al realizar los circuitos sin elementos de retardo, el retardo inherente en la red sólo se debe a los dispositivos lógicos y los cables.

Dividiremos los efectos del retardo sobre el desempeño de los circuitos en dos categorías. La primera incluye los efectos causados por los retardos en las rutas de retroalimentación; la segunda analiza los efectos de los retardos en la lógica y los cables.

Con frecuencia, utilizamos dispositivos de tipo inercial como elementos de retardo en las rutas de retroalimentación. Un *elemento de retardo inercial* es un elemento que sólo responde a señales que persisten durante un tiempo mayor o igual que el tiempo de retardo del dispositivo. Para ser más específicos, sea *ID* un retardo inercial con entrada Y, salida y y retardo Δt, como se muestra en la figura 10.44a. La salida y asume el valor de entrada Y después de un tiempo Δt si la duración de Y es mayor o igual que Δt. Una entrada de duración menor que Δt no se propagará a la salida. La figura 10.44b muestra la respuesta de un elemento de retardo inercial a una entrada típica.

Los elementos de retardo inercial sirven para filtrar los estados transitorios no deseados que se pueden presentar en las señales de retroalimentación. Estos estados transitorios se producen como resultado de los diversos tiempos de respuesta de los elementos lógicos y, si no son eliminados, pueden dar lugar a un comportamiento incorrecto en el circuito.

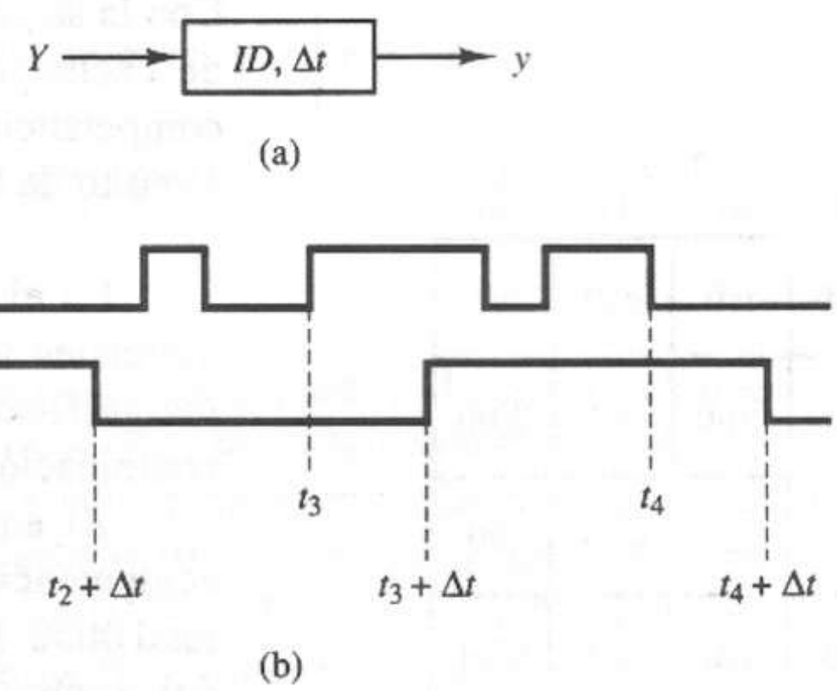


Figura 10.44 Elemento de retardo inercial. (a) Elemento de retardo inercial. (b) Secuencia típica de respuesta.

Los elementos de retardo inercial son más difíciles de realizar que los de retardo puro. Sin embargo, sus ventajas superan la complejidad de la realización. Por tanto, en el resto del capítulo supondremos que todos los elementos de retardo son inerciales.

10.6.1 Competencias y ciclos

Existe una *condición de competencia* en un circuito de modo fundamental si dos o más variables de estado secundario deben cambiar de valor cuando el circuito debe hacer una transición de un estado estable a otro estado estable. En los circuitos físicos, lo usual es que el retardo por las diversas rutas de retroalimentación no sea el mismo. Cuando es posible que existan retardos diferentes, una condición de competencia puede provocar un funcionamiento inesperado o incorrecto del circuito. Ilustraremos tal condición más adelante. Durante el resto del análisis sobre circuitos de modo fundamental, supondremos que los elementos de retardo en un ruta de retroalimentación no son los mismos que en otras.

Una condición de competencia *no es crítica* si el circuito opera siempre en forma adecuada en presencia de la competencia. El circuito opera en forma adecuada si termina en el estado estable correcto después de cualquier cambio de entrada. Sin embargo, en muchos casos, una competencia puede hacer que el circuito entre y permanezca en un estado estable incorrecto. Este último caso es una condición de competencia *crítica*. Al diseñar un circuito debemos evitar siempre las competencias críticas. Por otro lado, los diseñadores pueden con frecuencia sacar provecho de las competencias no críticas.

Con una asignación adecuada de estados secundarios, podemos evitar las condiciones de competencia crítica. Este problema de asignación no es trivial y lo analizaremos con detalle más adelante. Para comprender con mayor claridad los problemas de las condiciones de competencia, examinemos el siguiente ejemplo.

EJEMPLO 10.10

Considerar la tabla de flujo de la figura 10.45 como una vía para analizar las condiciones de competencia críticas y no críticas.

Con la asignación de estados $a = 00$, $b = 01$, $c = 10$ y $d = 11$ obtenemos la tabla de excitación de la figura 10.46. Esta tabla presenta ambas condiciones de competencia, críticas y no críticas. La figura 10.47 muestra una realización en circuito de la tabla.

	x_1x_2 = 00	01	11	10
a	ⓐ/0	b/0	ⓐ/1	b/1
b	a/0	ⓑ/0	c/0	ⓑ/0
c	a/1	ⓒ/1	ⓒ/0	d/0
d	a/0	c/0	a/1	ⓓ/1

Figura 10.45 Tabla de flujo.

En el siguiente análisis, supondremos que los elementos de retardo son inerciales y que los retardos en las compuertas lógicas y en los cables son insignificantes. El *estado total* es el valor del vector $x_1x_2y_1y_2$, que es la combinación de entrada x_1x_2 y del estado secundario y_1y_2.

Al analizar la tabla de excitación vemos que existe una condición de competencia cuando se realiza una transición del estado total 1011 al estado total 0000. Mostraremos que esta competencia no es crítica mediante un análisis del circuito para $\Delta_1 t > \Delta_2 t$ y para $\Delta_1 t < \Delta_2 t$. A fin de simplificar nuestro análisis, sea t_0 un instante en que el circuito se encuentra en el estado 1011, sea t_1 el instante en que cambia el estado de entrada de 10 a 00, sea t_2 el instante en que

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10	00	01	11	10
00	00	01	00	01	0	0	1	1
01	00	01	10	01	0	0	0	0
10	00	10	10	11	1	1	0	0
11	00	10	00	11	0	0	1	1
	$Y_1 Y_2$				z			

Figura 10.46 Tablas de excitación y de salida.

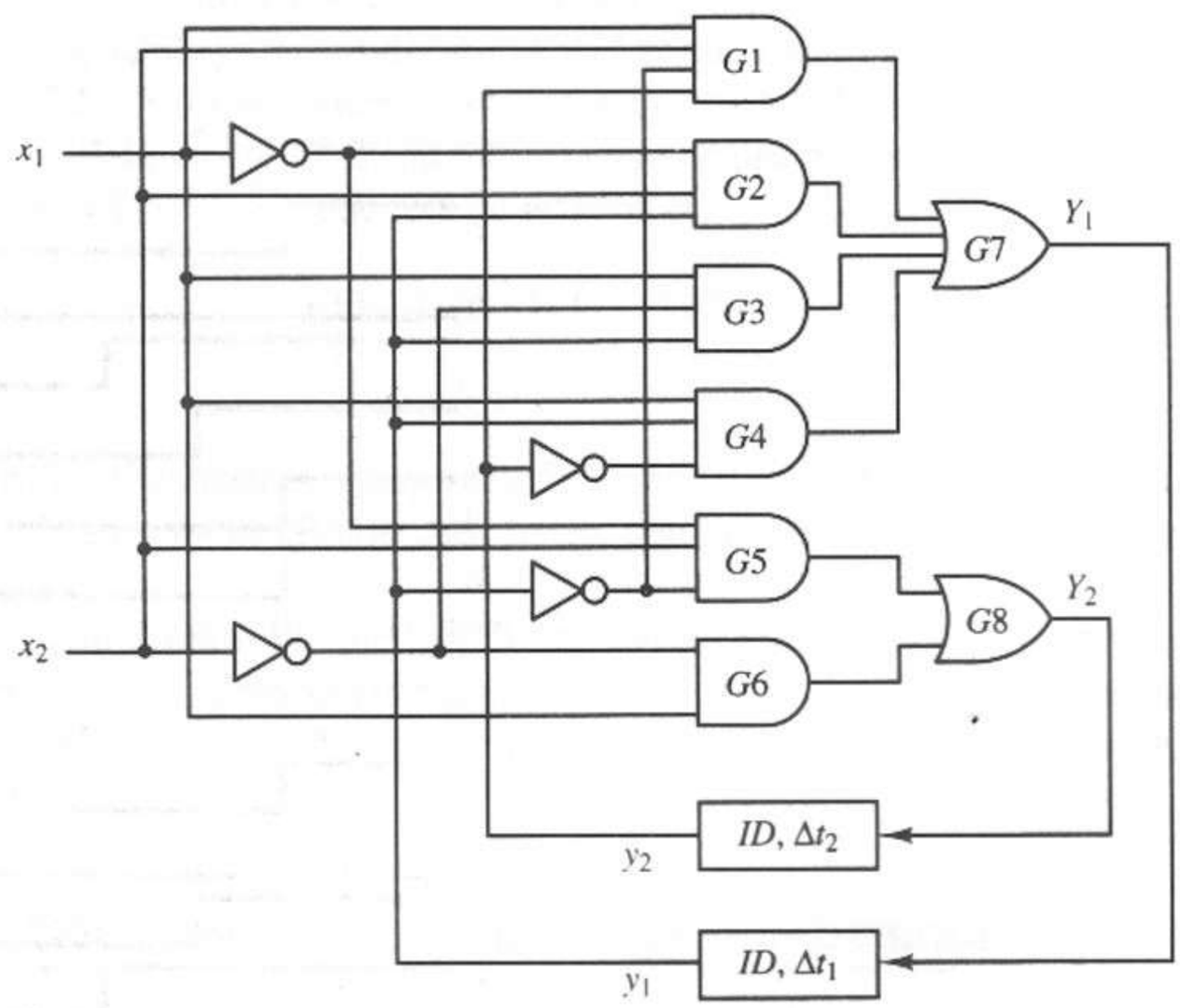


Figura 10.47 Circuito con competencias (por conveniencia, hemos omitido la lógica de salida).

responde el primer elemento de retardo, sea t_3 el instante en que responde el segundo elemento de retardo y sea t_4 un instante posterior a t_3.

Consideremos ahora la siguiente secuencia de eventos. En el instante t_0, el circuito se encuentra en el estado 1011 y las salidas de compuerta son $G1 = 0$, $G2 = 0$, $G3 = 1$, $G4 = 0$, $G5 = 0$, $G6 = 1$, $G7 = 1$ y $G8 = 1$. Cuando la entrada cambia de 10 a 00 en el instante t_1, todas las salidas de compuerta asumen el valor 0. Por tanto, $Y_1Y_2 = 00$. Sin embargo, y_1y_2 permanece con el valor 11 ya que son las salidas de los elementos de retardo; el circuito se encuentra en un estado inestable, pues $y_1 \neq Y_1$ y $y_2 \neq Y_2$.

En el resto del análisis influyen los tiempos de respuesta relativos de los elementos de retardo. Supongamos que $\Delta_1 t > \Delta_2 t$. En el instante $t_2 = t_1 + \Delta_2 t$, y_2 se convierte en 0 como respuesta a un cambio anterior en Y_2. $y_2 = 0$ no produce más cambios en Y_1 o Y_2. Por tanto, en $t_3 = t_1 + \Delta_1 t$, y_1 se convierte en 0 en respuesta a $Y_1 = 0$. Sin embargo, no hay un cambio en $Y_1 Y_2$ y el circuito alcanza un estado estable, ya que $y_1 = Y_1$ y $y_2 = Y_2$. La siguiente secuencia de estados totales es resultado de una competencia no crítica cuando $\Delta_1 t > \Delta_2 t$:

$$\underset{t_0}{1011}\text{–}\underset{t_1}{0011}\text{–}\underset{t_2}{0010}\text{–}\underset{t_3}{0000}$$

El lector debe verificar que obtenemos la siguiente secuencia cuando $\Delta_1 t < \Delta_2 t$:

$$\underset{t_0}{1011}\text{–}\underset{t_1}{0011}\text{–}\underset{t_2}{0001}\text{–}\underset{t_3}{0000}$$

La figura 10.48 muestra los diagramas de tiempos de estos cambios de estado.

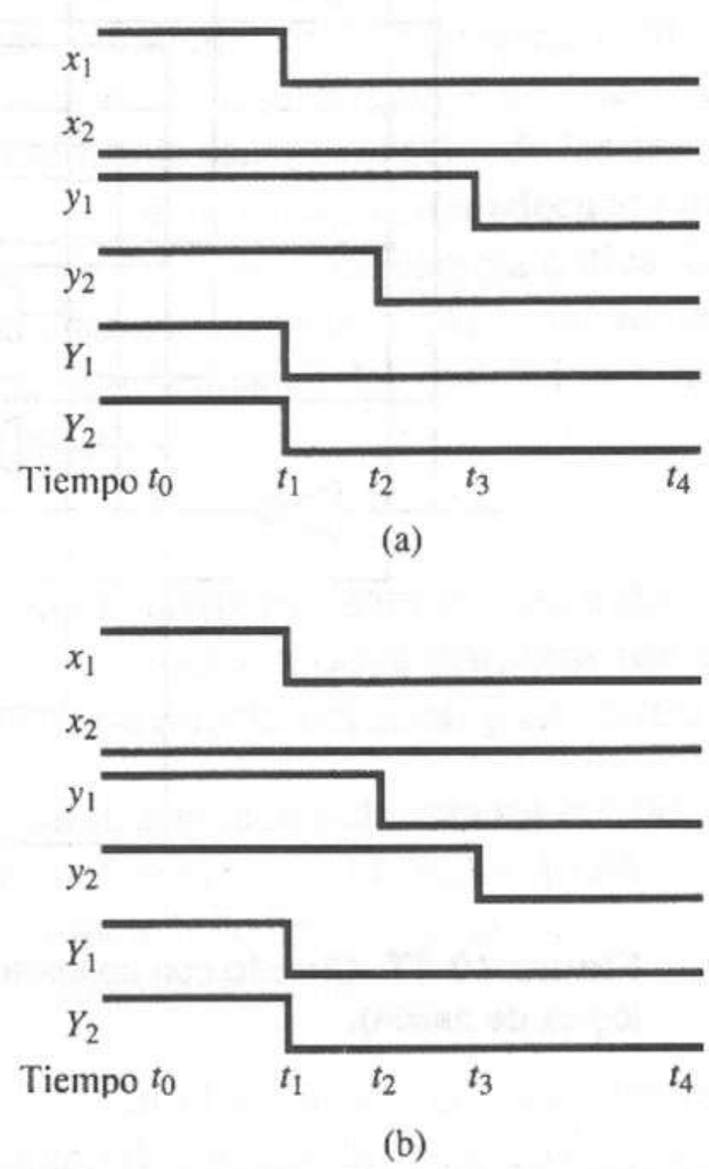


Figura 10.48 Tiempos para competencias no críticas. **(a)** $\Delta_1 t > \Delta_2 t$. **(b)** $\Delta_2 t > \Delta_1 t$.

Es importante observar que la respuesta del circuito fue una función de la relación entre $\Delta_1 t$ y $\Delta_2 t$. Sin embargo, en cada caso el estado estable final fue el estado deseado. Ésta es una característica de las competencias no críticas. Con frecuencia, la secuencia de estados inestables por los que pasa el circuito carece de importancia, siempre que se alcance el estado final correcto.

La ocurrencia de dos o más estados inestables consecutivos es un *ciclo*. Por ahora pospondremos nuestro análisis de este tema.

En contraste con las condiciones de competencia no críticas, una condición de competencia crítica puede conducir a un comportamiento erróneo del circuito. Por ejemplo, analicemos la respuesta del circuito si el estado total es 1001 y la entrada cambia de 10 a 11. Las compuertas del circuito tienen las siguientes salidas en el estado 1001: $G1 = 0$, $G2 = 0$, $G3 = 0$, $G4 = 0$, $G5 = 0$, $G6 = 1$, $G7 = 0$ y $G8 = 1$. En el instante t_1, cuando la entrada cambia a 11, $G1 = 1$ y $G6 = 0$, por lo que $Y_1 = 1$ y $Y_2 = 0$. La relación entre $\Delta_1 t$ y $\Delta_2 t$ influye en el resto del análisis. Supongamos ahora que $\Delta_1 t > \Delta_2 t$. En el instante $t_2 = t_1 + \Delta_2 t$, $y_2 = 0$, ya que $Y_2 = 0$. Sin embargo, cuando y_2 se convierte en 0 hace que $Y_1 = 0$ antes de que ID_1 responda a $Y_1 = 1$. Por tanto, $y_1 = 0 = Y_1$ y $y_2 = 0 = Y_2$, lo que implica que el dispositivo se ha estabilizado en el estado 1100. Ésta es una respuesta errónea, como lo indica la tabla de excitación. La secuencia de transición es

$$\begin{array}{cccc} 1001 & 1101 & 1100 & 1100 \\ t_0 & t_1 & t_2 & t_3 \end{array}$$

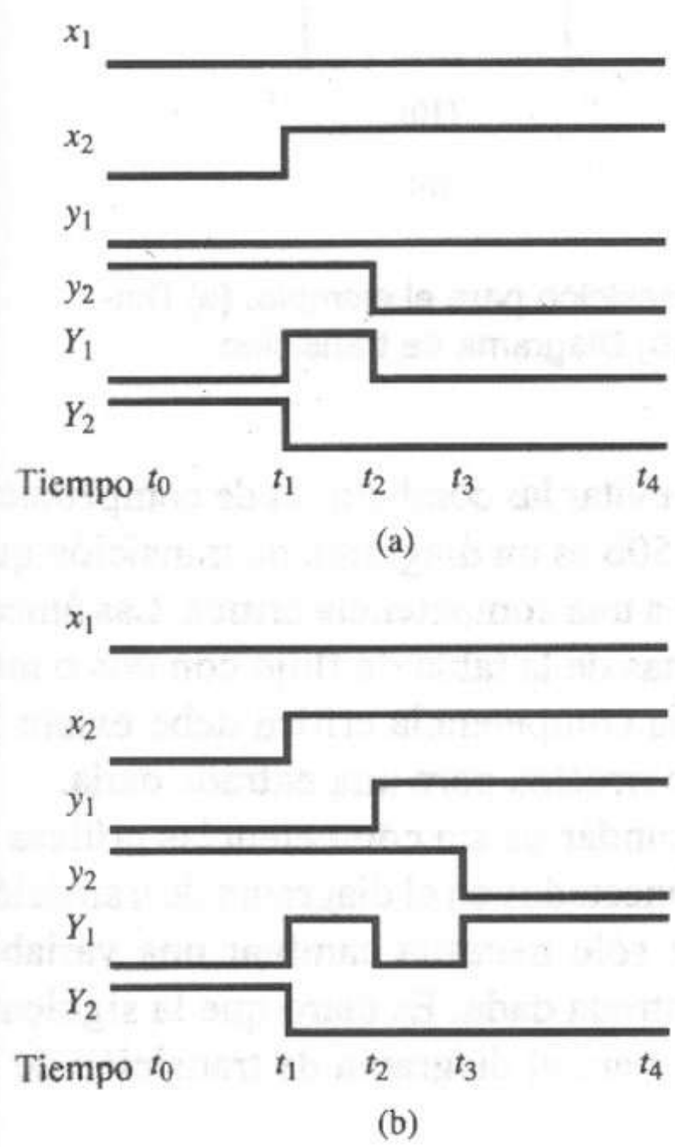


Figura 10.49 Competencia crítica. **(a)** $\Delta_1 t > \Delta_2 t$. **(b)** $\Delta_1 t < \Delta_2 t$.

Veamos ahora el caso $\Delta_1 t < \Delta_2 t$. Supongamos que $2\Delta_1 t > \Delta_2 t$. En el instante $t_2 = t_1 + \Delta_1 t$, y_1 se convierte en 1 en respuesta a $Y_1 = 1$. Este cambio obliga a que $G1 = 0$, lo que implica $Y_1 = 0$. En el instante $t_3 = t_1 + \Delta_2 t$, y_2 se convierte en 0, ya que $Y_2 = 0$. Cuando y_2 se convierte en 0, $G4$ se convierte en 1, haciendo de nuevo $Y_1 = 1$. Si $2\Delta_1 t > \Delta_2 t$, entonces $t_3 - t_2 < \Delta_1 t$. Por tanto, el cambio momentáneo en Y_1 no se refleja en y_1. El dispositivo se encuentra ahora en el estado estable especificado 1110. La secuencia de estados es:

$$\begin{array}{ccccc} 1001 & 1101 & 1111 & 1110 & 1110 \\ t_0 & t_1 & t_2 & t_3 & t_4 \end{array}$$

La figura 10.49 muestra un diagrama de tiempos más detallado. Recomendamos al lector buscar en el circuito las demás condiciones de competencia críticas existentes.

El ejemplo mostró que una condición de competencia crítica puede dar como resultado un comportamiento erróneo del circuito. Es claro que debemos evitar estas situaciones. Por tanto, ahora estudiaremos el problema de evitar las competencias críticas.

10.6.2 Cómo evitar condiciones de competencia

Podemos evitar las condiciones de competencia si elegimos en forma adecuada la asignación de estados secundarios. Dicho de manera sencilla, hay que asignar el estado secundario de modo que, en un instante dado, sólo tenga que cambiar una variable secundaria para cualquier transición de estado de la tabla de flujo. Con objeto de lograr esto, por lo general necesitamos establecer ciclos entre dos estados estables, así como incrementar el número de variables de estado utilizadas.

Ahora, consideremos el problema de lograr una asignación sin competencias para la tabla de flujo de la figura 10.45. Al analizar la tabla, vemos que debemos hacer transiciones de la fila *a* a la fila *b*, de *b* a *c*, de *c* a *d*, de *d* a *a* y de *c* a *a*. La figura 10.50a muestra el *diagrama de transición* que resume esta información. Cada nodo del diagrama corresponde a una fila de la tabla de flujo. Una línea une dos nodos si se pueden presentar transiciones entre las filas correspondientes. Rotulamos las líneas con los estados de entrada que pueden existir al presentarse la transición.

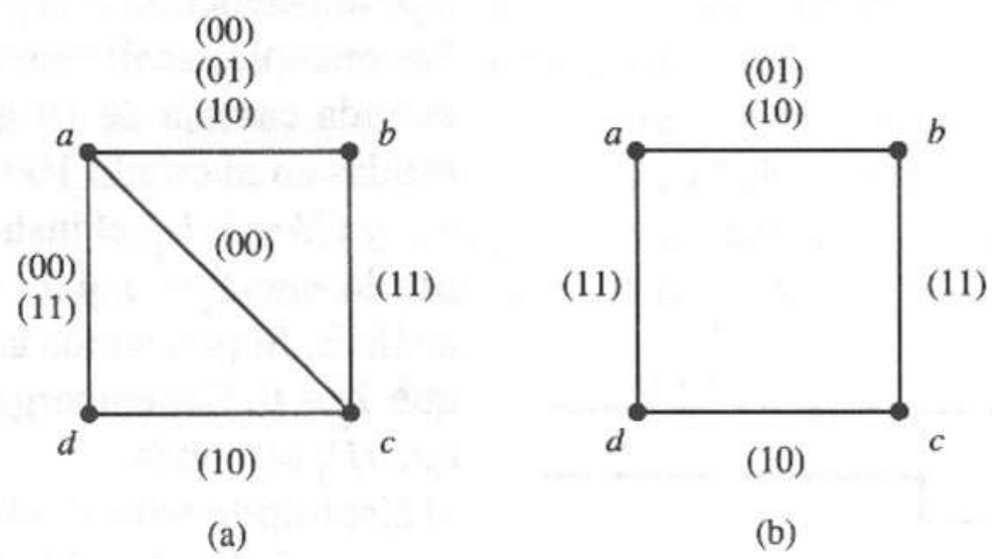


Figura 10.50 Diagrama de transición para el ejemplo. **(a)** Diagrama de transición completo. **(b)** Diagrama de transición crítica.

Como ya hemos dicho, sólo debemos evitar las condiciones de competencia críticas al diseñar el circuito. La figura 10.50b es un diagrama de transición que sólo incluye transiciones que pueden llevar a una competencia crítica. Las únicas transiciones críticas ocurren en las columnas de la tabla de flujo con dos o más estados estables, ya que para que haya una competencia crítica debe existir la posibilidad de llegar a estados estables incorrectos para una entrada dada.

Existe una asignación de estados secundarios sin competencias críticas si los códigos correspondientes a los nodos conectados en el diagrama de transición sólo difieren en 1 bit, lo que implica que sólo necesita cambiar una variable secundaria en respuesta a un cambio de entrada dada. Es claro que la siguiente asignación no tiene competencias críticas, para el diagrama de transición de la figura 10.50b.

Fila	$y_1\ y_2$
a	00
b	01
c	11
d	10

La figura 10.51 ilustra la tabla de excitación y la realización del circuito para esta asignación.

Este ejemplo tiene muchas otras asignaciones sin competencias críticas. Un procedimiento sencillo para elegir una asignación de este tipo consiste en codificar de manera arbitraria un estado, por ejemplo, $c = 10$; después, codificamos uno de los estados conectados cambiando 1 bit en el código anterior, por ejemplo, $b = 00$ y después repetimos este procedimiento hasta codificar todos los estados, por ejemplo, $a = 01$ y $d = 11$. Por regla general, el problema de asignación de estados es más complejo y lo analizaremos detalladamente en la siguiente sección.

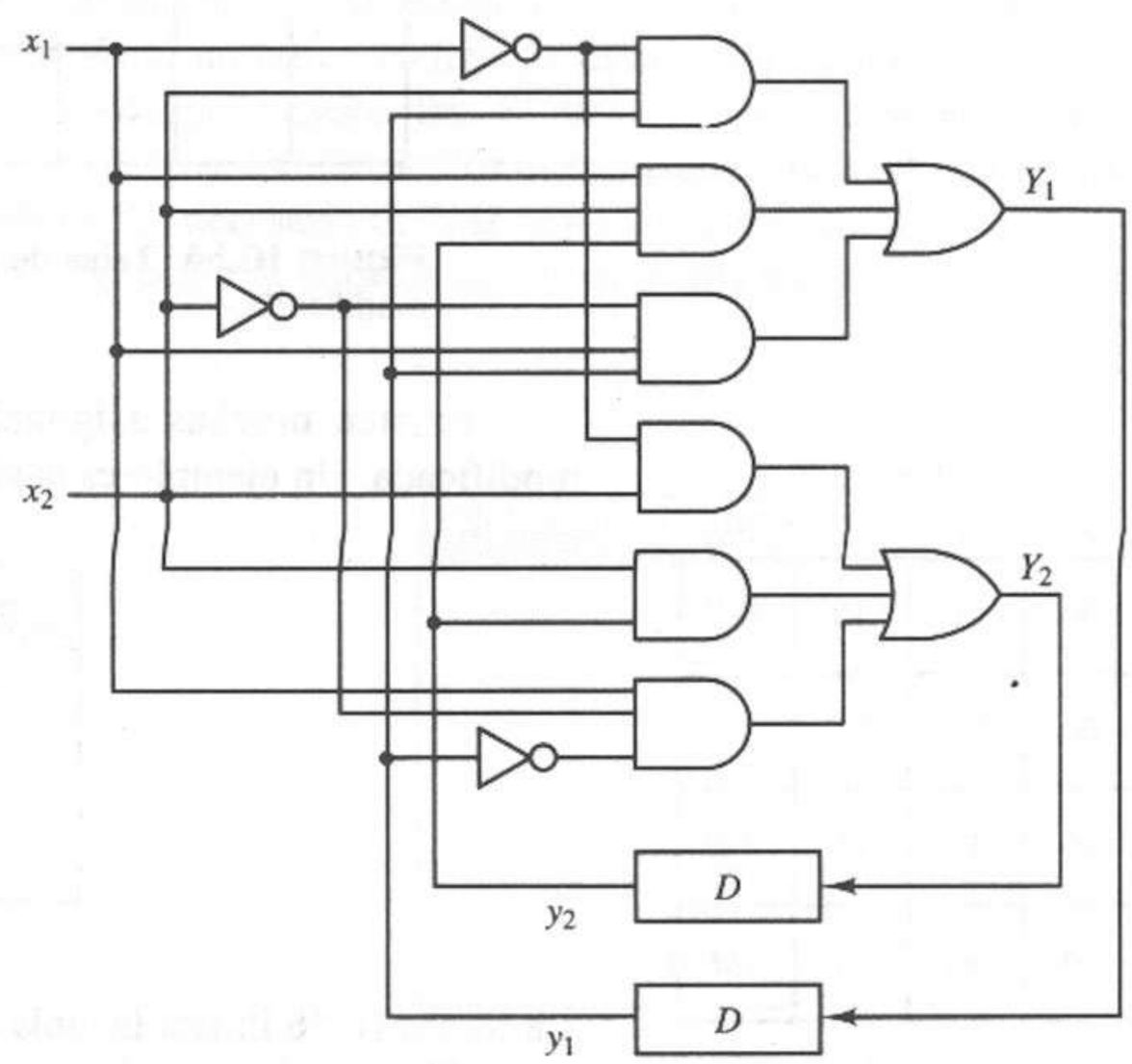
10.6.3 Asignaciones de estados libres de competencia

Describiremos ahora dos métodos para asignar estados sin competencia. El primero se basa en la creación de ciclos entre estados estables, mientras que en

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00	00	01	00	01
01	00	01	11	01
11	00	11	11	10
10	00	11	00	10

$Y_1 Y_2$

(a)



(b)

Figura 10.51 Realización sin competencias críticas. **(a)** Tabla de excitación. **(b)** Realización del circuito.

	00	01	11	10
a	a/0	b/0	c/–	a/0
b	a/0	b/0	b/0	c/–
c	a/–	c/1	c/1	c/1

Figura 10.52 Tabla de flujo reducida.

el segundo método debemos establecer filas redundantes en la tabla de flujo. Por lo general, las asignaciones más económicas se logran con el primer método, pero el segundo es más directo.

Método 1. Consideremos la tabla de flujo reducida de la figura 10.52. El diagrama de transición crítica de la figura 10.53 indica claramente que no existe una asignación que satisfaga las adyacencias necesarias.

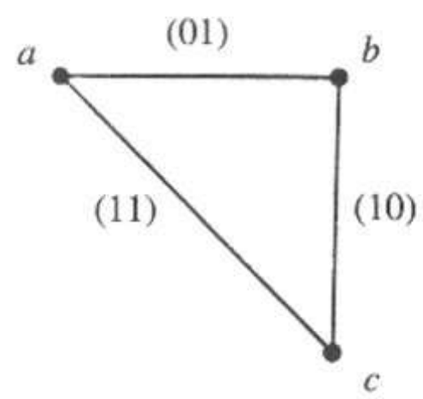


Figura 10.53 Diagrama de transición crítica.

Sin embargo, si modificamos la tabla de flujo de modo que exista un ciclo entre cualesquiera dos estados estables, podremos tener una asignación sin competencia. Para ilustrar este método, estableceremos un ciclo entre los estados a y c en la columna 11. La figura 10.54 muestra la tabla de flujo modificada.

Observe que se presenta la siguiente secuencia de estados durante la transición de (a) en la columna 10 a (c) en la columna 11:

$$(a) \rightarrow d \rightarrow c \rightarrow (c)$$

Por tanto, el diagrama de transición crítica queda como en la figura 10.55.

	x_1x_2 00	01	11	10
a	(a)/0	b/0	d/–	(a)/0
b	a/0	(b)/0	(b)/0	c/–
c	a/–	(c)/1	(c)/1	(c)/1
d	a/–		c/–	

Figura 10.54 Tabla de flujo modificada.

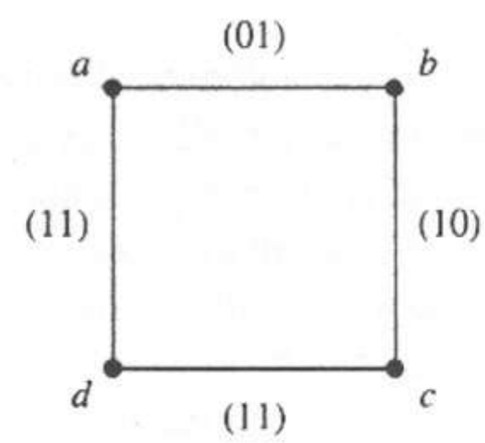


Figura 10.55 Nuevo diagrama de transición crítica.

Existen muchas asignaciones sin competencia para la tabla de flujo modificada. Un ejemplo es la siguiente.

Fila	y_1y_2
a	00
b	01
c	11
d	10

y_1y_2 \ x_1x_2	00	01	11	10
00	(00)	01	10	(00)
01	00	(01)	(01)	11
11	00	(11)	(11)	(11)
10	00	dd	11	dd

Y_1Y_2

Figura 10.56 Tabla de excitación.

La figura 10.56 ilustra la tabla de excitación resultante.

Observe el estado de excitación 00 asignado al estado 0010. Esta asignación evita la posibilidad de un estado no deseado en la fila 10.

En el ejemplo anterior creamos un ciclo sin tener que incrementar el número de variables de estado por encima del mínimo requerido. Esto no siempre es posible, como ahora veremos con la tabla de flujo de la figura 10.57. El diagrama de transición crítica de la figura 10.58 indica que no podemos cumplir con todas las adyacencias requeridas. En este caso sólo podremos crear ciclos para obtener asignaciones de estado sin competencia si utilizamos tres variables de estado secundarias.

Hay varias formas de establecer ciclos en este problema que eviten competencias críticas. Una consiste en crear un ciclo entre b y c en la columna 11, un ciclo entre b y d en la columna 01 y un ciclo entre c y d en las columnas 00 y 10.

	x_1x_2 00	01	11	10
a	ⓐ/1	c/0	ⓐ/0	b/0
b	a/1	ⓑ/1	c/1	ⓑ/0
c	d/0	ⓒ/0	ⓒ/1	d/1
d	ⓓ/0	b/–	a/–	ⓓ/1

Figura 10.57 Ejemplo de tabla de flujo.

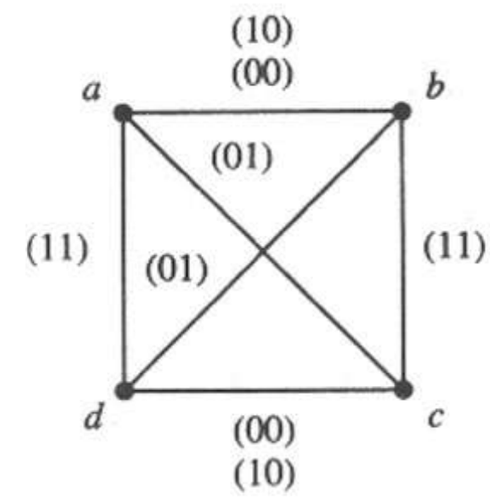


Figura 10.58 Diagrama de transición crítica.

La tabla de flujo modificada de la figura 10.59 ilustra estos ciclos. El diagrama de transición para esta tabla de flujo aparece en la figura 10.60.

Los mapas de adyacencias son útiles para elegir los códigos que satisfacen los requisitos dados en el diagrama de transición. En el capítulo 9 utilizamos esta técnica. El mapa tiene un formato similar al mapa K, excepto que cada celda representa un único código de estado. Por tanto, las celdas adyacentes representan códigos adyacentes. El mapa de tres variables de la figura 10.61 muestra una asignación de estados que satisface todas las adyacencias. Observe que también fue necesario utilizar otra transición de d a b.

La asignación de estados correspondiente es:

	x_1x_2 00	01	11	10
a	ⓐ/1	c/0	ⓐ/0	b/0
b	a/1	ⓑ/1	b^1/1	ⓑ/0
b^1	–/–	–/–	c/1	–/–
c	c^1/0	ⓒ/0	ⓒ/1	c^1/1
c^1	d/0	–/–	–/–	d/1
d	ⓓ/0	d^1/–	a/–	ⓓ/1
d^1	–/–	b/–	–/–	–/–

Figura 10.59 Tabla de flujo modificada con ciclos creados.

Fila	y_1 y_2
a	000
b	001
b^1	011
c	010
c^1	110
d	100
d^1	101

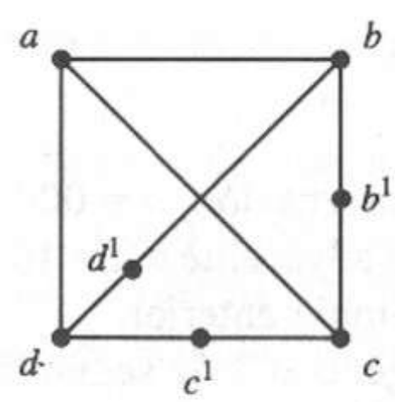


Figura 10.60 Diagrama de transición para la tabla de flujo modificada.

y_1	y_2y_3 00	01	11	10
0	a	b	b^1	c
1	d	d^1	–	c^1

Figura 10.61 Asignación de estados que satisfacen las adyacencias requeridas.

Ahora podemos obtener la tabla de excitación, misma que aparece en la figura 10.62.

$y_1 y_2 y_3$ \ $x_1 x_2$	00	01	11	10
000	(000)	010	(000)	001
001	000	(001)	011	(001)
011	*ddd*	*ddd*	010	*ddd*
010	110	(010)	(010)	110
110	100	*ddd*	*ddd*	100
100	(100)	101	000	(100)
101	*ddd*	001	*ddd*	*ddd*

$Y_1 Y_2 Y_3$

Figura 10.62 Tabla de excitación obtenida con el método 1.

	$y_1 y_2 y_3$ \ $x_1 x_2$	00	01	11	10
a	000	(000)	010	(000)	001
a	100	(100)	000	(100)	000
b	001	000	(001)	011	(001)
b	011	001	(011)	010	(011)
c	010	110	(010)	(010)	110
c	110	111	(110)	(110)	111
d	101	(101)	001	100	(101)
d	111	(111)	101	101	(111)

$Y_1 Y_2 Y_3$

Figura 10.63 Tabla de excitación obtenida con el método 2.

Método 2. Este método se basa en la duplicación de filas de la tabla de flujo reducida. Asignamos estados a la tabla desarrollada de modo que una fila de cada conjunto de filas equivalentes sea adyacente a una fila de cada conjunto restante de filas equivalentes. Además, cada fila dentro de un conjunto de filas equivalentes es adyacente al menos a otra fila del mismo conjunto. Por tanto, podemos lograr transiciones sin competencia entre dos estados estables cualesquiera estableciendo en forma adecuada las transiciones de una fila a otra.

En el caso de las tablas de flujo de cuatro filas, este método implica la duplicación de cada fila. La tabla siguiente muestra las asignaciones de estado para la tabla desarrollada:

$y_2 y_3$ \ y_1	0	1
00	*a*	*a*
01	*b*	*d*
11	*b*	*d*
10	*c*	*c*

Para esta asignación, $a = 000$ es adyacente a $b = 001$ y $c = 010$, mientras que $a = 100$ es adyacente a $d = 101$. La figura 10.63 muestra la tabla de excitación para el ejemplo anterior.

En general, este segundo método no es tan económico como el primero, pues no existen condiciones prescindibles en la tabla de excitación final. Sin embargo, el método no requiere una elección de código mediante prueba y error, ya que se han publicado códigos para tablas de diversos tamaños. La figura 10.64 ilustra las tablas de asignación para tablas de seis y ocho filas.

$y_3 y_4$ \ $y_1 y_2$	00	01	11	10
00	e	b	e	e
01	e	b	f	f
11	c	c	e	f
10	d	d	e	f

(a)

$y_3 y_4 y_5$ \ $y_1 y_2$	00	01	11	10
000	a	b	c	c
001	a	b	d	d
011	a	a	c	d
010	b	b	c	d
110	e	f	e	e
111	e	f	f	f
101	g	g	g	h
100	h	h	g	h

(b)

Figura 10.64 Tablas de asignación para seis y ocho filas. **(a)** Tablas de seis filas. **(b)** Tablas de ocho filas.

EJEMPLO 10.11

Diseñar un circuito de modo fundamental de dos entradas (x_1,x_2) y una salida (z) que opere como sigue. La salida cambia de 0 a 1 sólo en el primer cambio de entrada x_1 posterior a un cambio de entrada x_2. Hay un cambio de salida de 1 a 0 sólo cuando x_1 cambia de 1 a 0 mientras x_2 = 1.

	$x_1 x_2$ 00	01	11	10
1	1/0	2/–	–/–	4/–
2	1/–	2/0	3/–	–/–
3	–/–	5/–	3/1	4/–
4	6/–	–/–	3/–	4/1
5	1/–	5/0	7/–	–/–
6	6/1	8/–	–/–	4/–
7	–/–	5/–	7/0	9/–
8	6/–	8/1	3/–	–/–
9	6/–	–/–	10/–	9/0
10	–/–	8/–	10/0	9/–
11	11/0	2/–	–/–	12/–
12	11/–	–/–	7/–	12/0

Figura 10.65 Tabla de flujo primitiva para el ejemplo.

Paso 1. La tabla de flujo primitiva de la figura 10.65 satisface los requisitos establecidos. Observe la presencia de dos estados con salida 0 en cada columna. Esta condición es consecuencia del requisito de que un cambio de 0 a 1 en la salida sólo se presenta en el primer cambio de entrada x_1 posterior a un cambio de entrada x_2.

Paso 2. El uso de una tabla de implicación para la tabla de flujo primitiva indica que (1,2), (3,4), (4,6), (4,8), (5,7), (6,8), (9,10) y (11,12) son filas compatibles. La figura 10.66 muestra el diagrama de fusión correspondiente.

Como lo indica el diagrama, podemos fusionar las filas 4, 6 y 8 en una sola. Las demás filas deben unirse por pares. La figura 10.67 muestra la tabla de flujo reducida mínima obtenida del diagrama de fusión, donde $a = (1,2)$, $b = (3,4)$, $c = (5,7)$, $d = (4,6,8)$, $e = (9,10)$ y $f = (11,12)$. Observe que hemos completado la asignación de salida que aparece en la tabla de flujo reducida a fin de evitar un *glitch* de salida, como se describió en los puntos 2 y 3 del paso 1 del ejemplo 10.7.

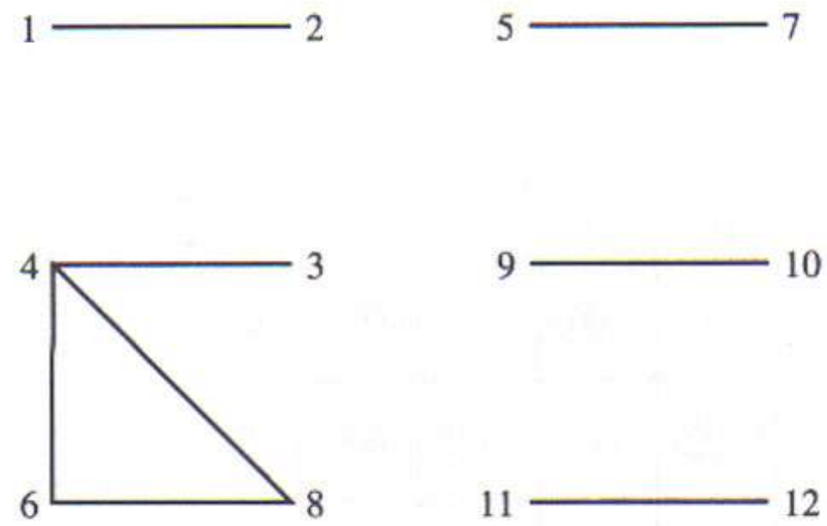


Figura 10.66 Diagrama de fusión.

	x_1x_2 00	01	11	10
a	$a/0$	$a/0$	$b/-$	$b/-$
b	$d/1$	$c/-$	$b/1$	$b/1$
c	$a/0$	$c/0$	$c/0$	$e/0$
d	$d/1$	$d/1$	$b/1$	$d/1$
e	$d/-$	$d/-$	$e/0$	$e/0$
f	$f/0$	$a/0$	$c/0$	$f/0$

Figura 10.67 Tabla de flujo reducida mínima.

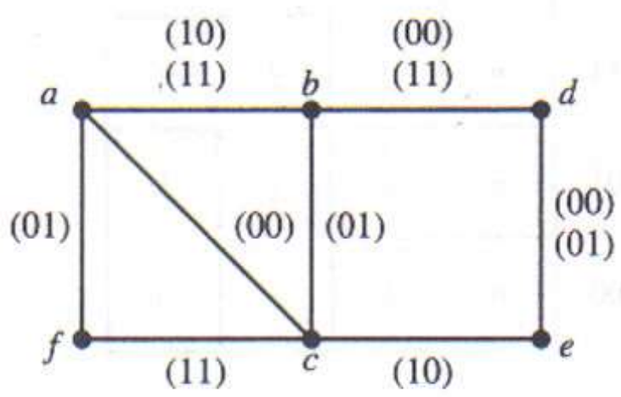


Figura 10.68 Diagrama de transición crítica.

Paso 3. La figura 10.68 muestra el diagrama de transición crítica. Se requiere un mínimo de tres variables de estados secundarios, ya que la tabla de flujo reducida tiene seis filas. Podemos obtener una asignación de tres variables aprovechando de manera inteligente los ciclos.

Utilizaremos el método 1 para obtener una asignación sin competencias. Los estados intermedios c^1 y f^1, agregados como en la figura 10.69, permiten hacer la asignación. El mapa de la figura 10.70 ilustra la asignación elegida.

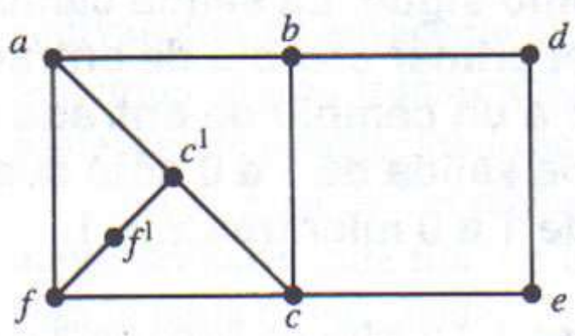


Figura 10.69 Asignación de estados obtenida con el método 1.

	y_1	y_2y_3 00	01	11	10
	0	a	b	c	c^1
	1	f	d	e	f^1

Figura 10.70 Mapa de la asignación de estados elegida.

Paso 4. La figura 10.71 muestra las tablas de excitación y de salida.

Paso 5. Las ecuaciones de excitación y de salida correspondientes son

$$Y_1 = \bar{x}_1\bar{x}_2\bar{y}_2y_3 + x_1\bar{x}_2y_2 + y_1y_2y_3 + x_1y_1\bar{y}_2\bar{y}_3 + \bar{x}_1y_1y_3 + \bar{x}_2y_1$$

$$Y_2 = \bar{x}_1x_2\bar{y}_1y_3 + x_1x_2y_1\bar{y}_3 + \bar{y}_1y_2y_3 + x_1y_2$$

$$Y_3 = \bar{y}_2y_3 + x_1\bar{x}_2\bar{y}_1 + y_1y_3 + x_2y_3$$

$$z = \bar{y}_2y_3$$

Paso 6. Una realización lógica de estas ecuaciones completa el proceso de diseño.

$y_1y_2y_3$	x_1x_2 00	01	11	10	x_1x_2 00	01	11	10
a 000	000	000	001	001	0	0	–	–
b 001	101	011	001	001	1	–	1	1
c^1 010	000	–	011	–	0	–	0	–
c 011	010	011	011	111	0	0	0	0
f 100	100	000	110	100	0	0	0	0
d 101	101	101	001	101	1	1	1	1
f^1 110	–	–	010	–	–	–	0	–
e 111	101	101	111	111	–	–	0	0
	$Y_1Y_2Y_3$				z			

Figura 10.71 Tablas de excitación y de salida.

10.6.4 Riesgos

En el capítulo 3 iniciamos el análisis del tema de los riesgos para las redes lógicas combinatorias. Los riesgos estáticos y dinámicos también pueden aparecer en la parte lógica combinatoria de los circuitos secuenciales y deben tomarse en cuenta al diseñar circuitos secuenciales. El análisis anterior sigue siendo válido en este caso, por lo que no trataremos más este tema. Sin embargo, es importante señalar que con frecuencia se utilizan elementos de retardo inercial para filtrar los estados transitorios provocados por estos riesgos.

Existe un tercer tipo de riesgos que sólo se presenta en los circuitos de modo fundamental y lo analizaremos aquí brevemente. En el siguiente análisis, supondremos que todos los elementos lógicos tienen cierto retardo inherente asociado. Un *riesgo esencial* es un riesgo causado por retardos desiguales a través de dos o más rutas que parten de la misma línea de entrada. Un riesgo así puede hacer que el circuito responda de manera incorrecta a los cambios de entrada. Para ilustrar esta situación, consideremos el circuito de la figura 10.72a. La figura 10.72b muestra la tabla de excitación y de salida para el circuito.

Supongamos que el circuito se encuentra en el estado $x = y_1 = y_2 = 0$. Por tanto, $Y_1 = Y_2 = 0$. Además, supongamos que la compuerta NOT $N1$ tiene un retardo asociado muy grande comparado con los retardos de los demás elementos del circuito, incluido el retardo de retroalimentación. Consideremos ahora la respuesta del circuito a un cambio de 0 a 1 en x en el instante t_1. La figura 10.72c muestra un diagrama de tiempos de la respuesta. Como puede verse, el circuito se estabiliza en el estado $x = 1, y_1 = 1$ y $y_2 = 0$. Esto es una respuesta incorrecta, según lo muestra la tabla de excitación.

Los eventos críticos ocurren en los instantes t_5, t_6, t_{10} y t_{13}. El circuito se encuentra en el estado secundario correcto 01 en el instante t_5. Sin embargo, como $N1$ no ha respondido aún al cambio de entrada, $A2$ cambia a 1, lo que hace que $Y_1 = 1$ en el instante t_6. Esto hace a su vez que $A3 = 0$. En t_{10}, $N1$ cambia a 0,

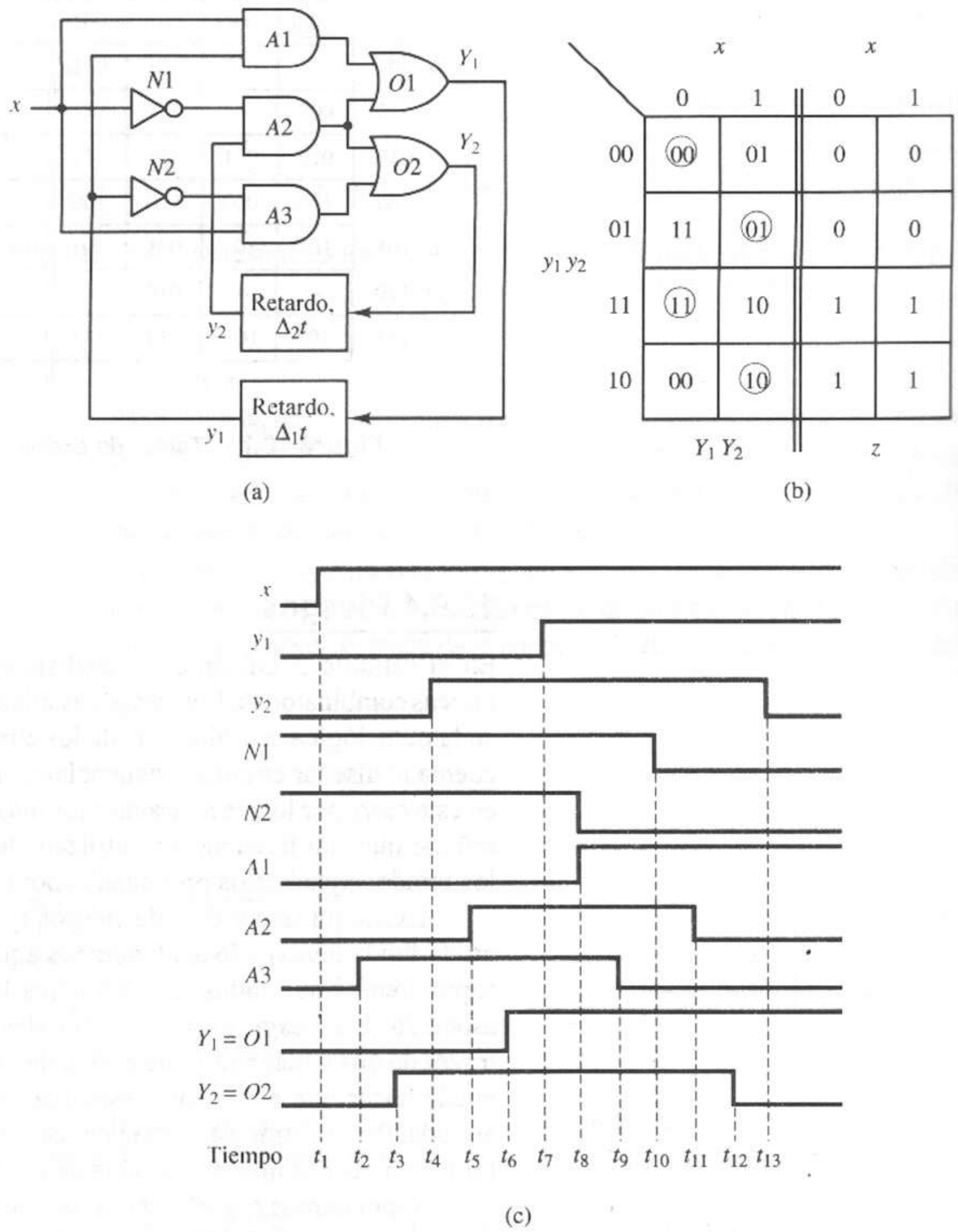


Figura 10.72 Circuito con riesgo esencial. **(a)** Diagrama de circuito. **(b)** Tablas de excitación y de salida. **(c)** Diagrama de tiempos.

haciendo que $A2 = 0$, lo que implica $Y_2 = 0$. En t_{13}, y_2 responde a $Y_2 = 0$ y el circuito llega a una condición estable.

Por tanto, el retardo en $N1$ ha obligado incorrectamente a que $Y_1 = 1$ en el instante t_6, lo cual dispara la secuencia de eventos que conducen a un estado estable incorrecto. Podemos superar el efecto de tales retardos proporcionando un retardo suficiente en las rutas de retroalimentación.

10.6.5 Análisis

El procedimiento de análisis dado en este capítulo implicó determinar las tablas de excitación, de salida y de flujo a partir de un diagrama de circuito. Por lo general, estas tablas ameritan un estudio detallado para determinar si existe una competencia crítica o un riesgo en el circuito.

10.7 Resumen

Este capítulo fue una introducción al tema de los circuitos secuenciales asíncronos. Examinamos circuitos de modo pulso y de modo fundamental. Primero centramos nuestra atención en el análisis y diseño de los circuitos de modo pulso. Presentamos los procedimientos de análisis y diseño y los ilustramos con diversos ejemplos. A continuación examinamos el análisis y diseño de los circuitos de modo fundamental. También establecimos los procedimientos de análisis y diseño y los demostramos mediante ejemplos. Por último, realizamos un análisis de las competencias y los riesgos. Presentamos e ilustramos los procedimientos para asignar estados libres de competencia.

REFERENCIAS BIBLIOGRÁFICAS

1. S. H. CALDWELL, *Switching Circuits and Logical Design*. New York: Wiley, 1958.
2. D. A. HUFFMAN, "The Synthesis of Sequential Switching Circuits," *J. Franklin Institute*, Vol. 257, No. 3, Mar. 1954, pp. 161–190, and No. 4, Apr. 1954, pp. 275–303.
3. D. A. HUFFMAN, "A Study of Memory Requirements of Sequential Switching Circuits," *Tech. Report No. 293*, Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA, Mar. 14, 1955.
4. E. J. MCCLUSKEY, "Fundamental and Pulse Mode Sequential Circuits," *IFIP Congress Proc. 1962*. Amsterdam: North-Holland Publishing Company, 1963.
5. S. H. UNGER, *Asynchronous Sequential Switching Circuits*. New York: Wiley (Interscience Division), 1969.

PROBLEMAS

10.1 Analice el circuito de modo pulso de la figura P10.1.

(a) Determine una tabla de estados.

(b) Construya un diagrama de tiempos para el circuito en respuesta a la siguiente secuencia de entrada. Incluya x_1, x_2, x_3, y_1, y_2, J_1, K_1, J_2, K_2, Y_1, Y_2 y z en su diagrama.

10.2 Analice el circuito de modo pulso de la figura P10.2.

(a) Determine una tabla de estados.

(b) Determine la respuesta de salida a la secuencia de entrada x_1–x_2–x_1–x_1–x_1–x_1–x_2–x_2 si el estado inicial es 00.

(c) ¿Qué forma (nivel o pulso) tendrá una salida de $z = 1$? ¿Por qué?

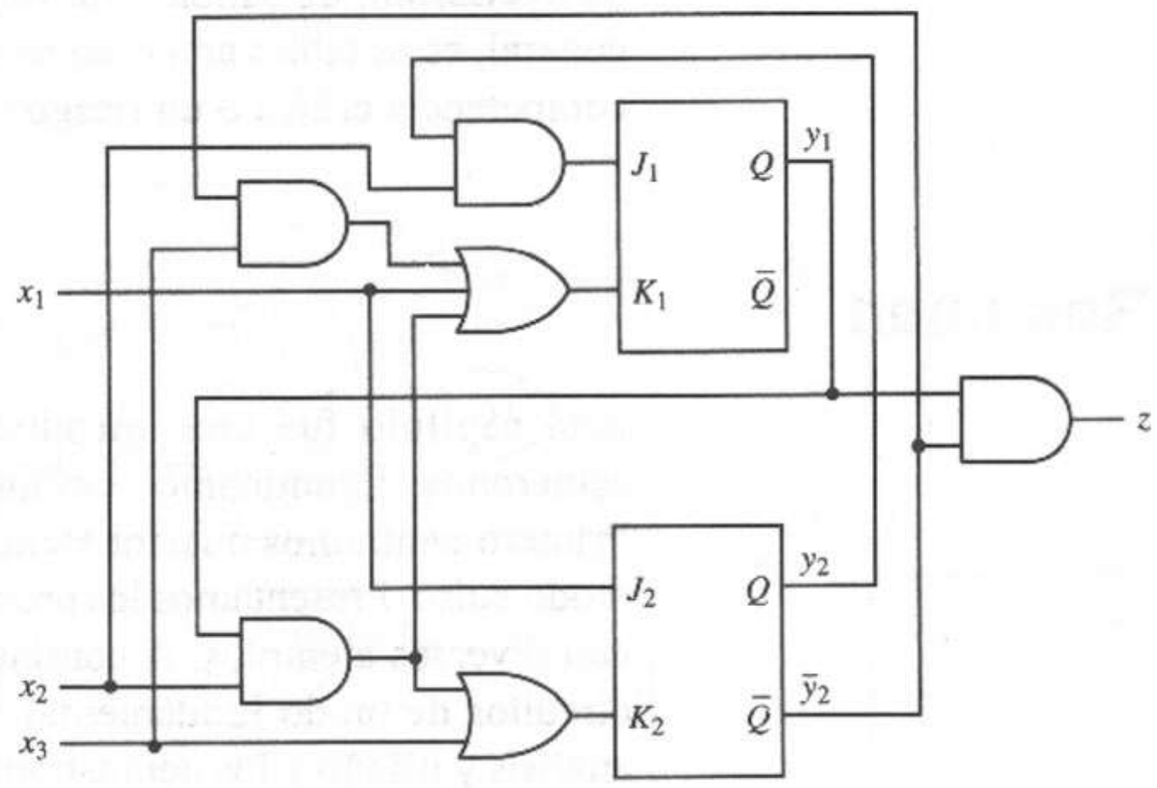


Figura P10.1

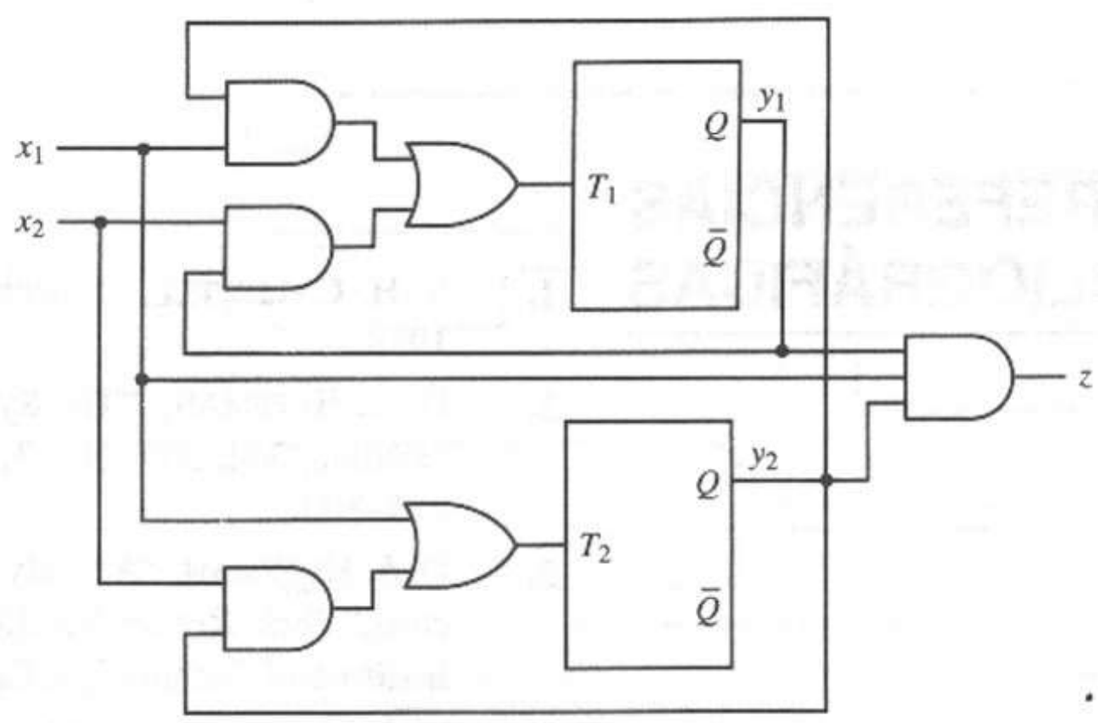


Figura P10.2

10.3 Determine una realización de la siguiente tabla de estados de modo pulso. Utilice *flip-flops* JK con compuertas AND, OR y NOT.

Estado actual	x_1	x_2	x_3
A	$A/0$	$B/0$	$C/1$
B	$B/0$	$C/0$	$D/0$
C	$C/0$	$D/0$	$A/1$
D	$D/0$	$A/0$	$B/1$

Estado siguiente/z

10.4 Diseñe un circuito de modo pulso que cumpla lo siguiente. Utilice compuertas AND, OR y NOT con *flip-flops* SR para realizar el circuito. El circuito tendrá dos entradas x_1 y x_2 y una salida z. Producirá un pulso de salida en forma simultánea

con el último de una serie de tres pulsos de entrada si y sólo si la secuencia contenía al menos dos pulsos x_1.

10.5 Se necesita un circuito secuencial de modo pulso que satisfaga lo siguiente: debe tener dos líneas de entrada x_1 y x_2 y una línea de salida z. Producirá una transición de salida de 0 a 1 sólo en el último pulso x_2 de la secuencia x_1–x_2–x_1–x_2. La salida va de 0 a 1 sólo en el primer pulso x_1 que ocurre después de la transición de salida de 0 a 1. Permita secuencias traslapadas. Diseñe el circuito mediante *flip-flops* T con compuertas AND, OR y NOT.

10.6 Analice el circuito de modo fundamental de la figura P10.6.

(a) Elabore las tablas de excitación y de salida.

(b) Construya una tabla de flujo.

(c) Utilice la tabla de flujo para determinar la respuesta de salida a la secuencia de entrada x_1x_2: 00–01–11–10–00–01–00–10. Suponga que, inicialmente, $x_1 = x_2 = y_1 = y_2 = Y_1 = Y_2 = 0$.

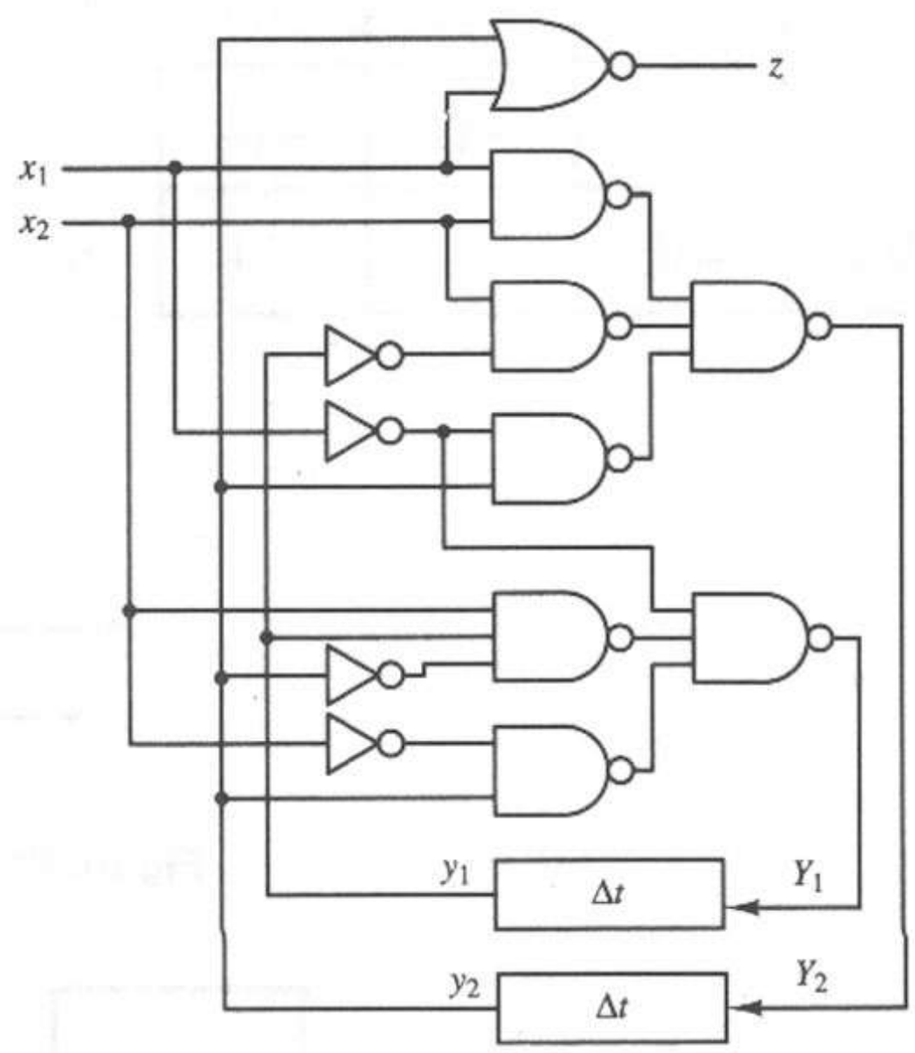


Figura P10.6

10.7 Considere el circuito de la figura P10.7a. Analice el circuito como sigue:

(a) Construya un diagrama de tiempos para la secuencia de entrada de la figura P10.7b. Suponga que no hay retardo en las compuertas lógicas. Suponga además que en un principio $y_1 = Y_1 = 1$ y $y_2 = Y_2 = 0$. Incluya x_1, x_2, y_1, y_2, Y_1, Y_2 y z en el diagrama de tiempos.

(b) Repita la parte (a) suponiendo que cada compuerta lógica tiene un retardo de ½ ΔT.

10.8 Prepare una tabla de flujo primitiva para un circuito de modo fundamental que cumpla lo siguiente. Se necesita una entrada x y una salida z. La salida debe seguir a la entrada cada segunda transición 0–1–0, como lo indica la figura P10.8.

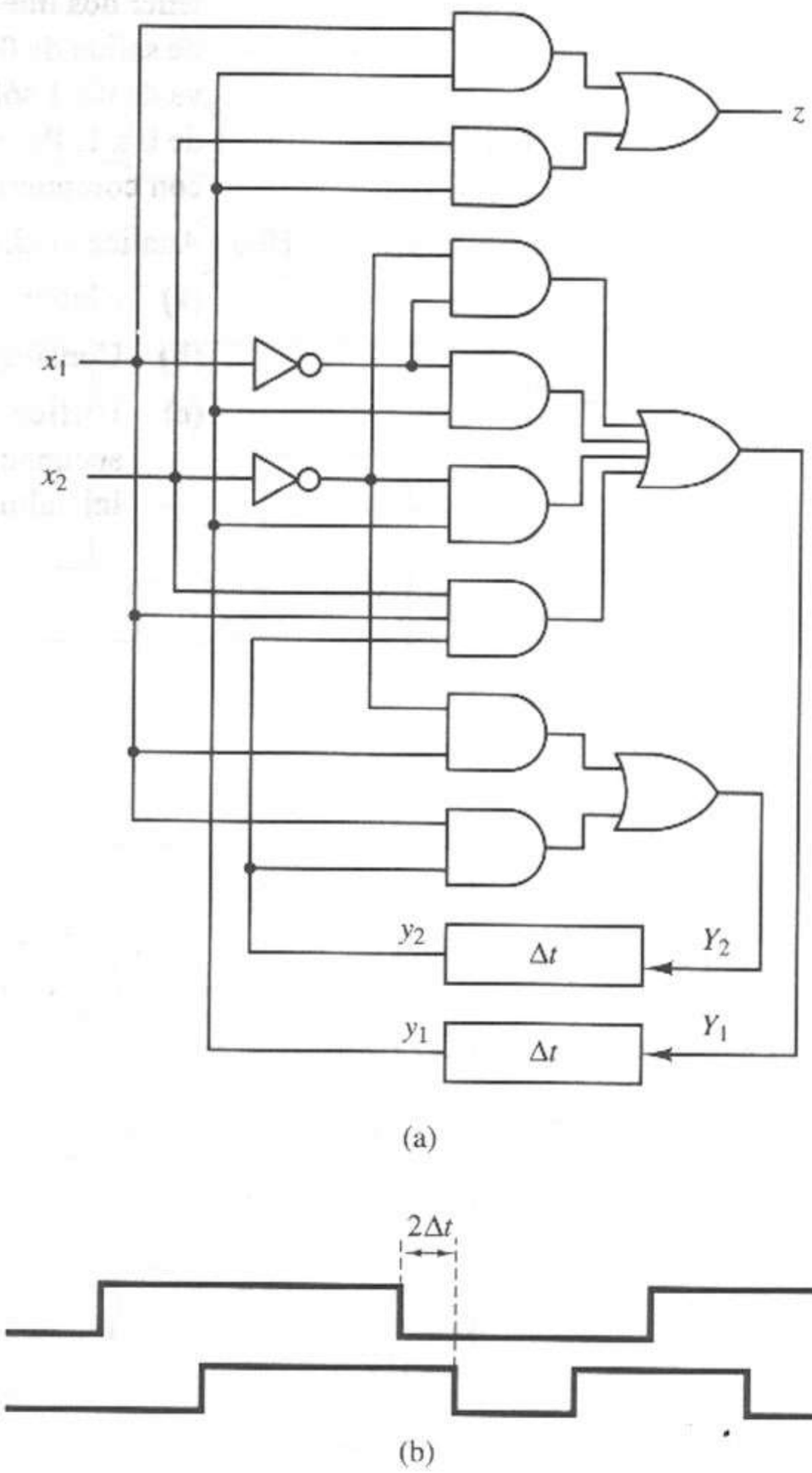


Figura P10.7 **(a)** Diagrama lógico. **(b)** Secuencia de entrada.

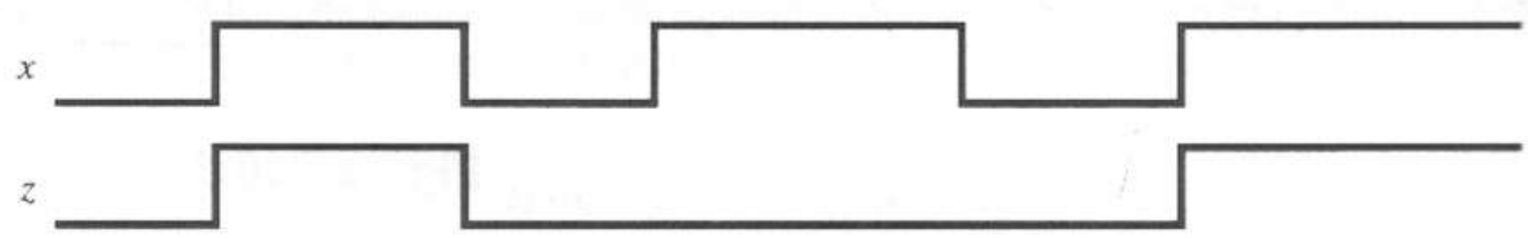


Figura P10.8

10.9 Hay que diseñar un circuito de modo fundamental que satisfaga lo siguiente. Se necesitan dos entradas (x_1,x_2) y una salida (z). La salida $z = 0$ se produce siempre que $x_1 = x_2$. Cuando $x_1 = 0$ y x_2 cambia de 0 a 1, debe ocurrir una salida $z = 1$. Cuando $x_1 = 1$ y x_2 cambia de 1 a 0, debe ocurrir una salida $z = 1$. Fuera de esto ningún cambio de entrada causará un cambio de salida. Prepare una tabla de flujo primitiva para el circuito.

10.10 Construya una tabla de flujo primitiva para un circuito de modo fundamental con las siguientes especificaciones. El circuito debe tener dos entradas (x_1,x_2) y dos salidas (z_1,z_2). Cuando $x_1 = x_2 = 0$, las salidas deben ser $z_1 = z_2 = 0$. Si $x_2 = 1$ y x_2 cambia de 0 a 1, debemos obtener una salida $z_1 = 0$, $z_2 = 1$. Si $x_2 = 1$ y x_1 cambia de 0 a 1, debemos obtener una salida $z_1 = 1$, $z_2 = 0$. Las salidas se reestablecen a $z_1 = z_2 = 0$ sólo si $x_1 = x_2 = 0$. No se produce un cambio de salida en los demás cambios de entrada.

10.11 Reduzca la siguiente tabla de flujo primitiva a una tabla con un mínimo de filas:

	x_1x_2 00	01	11	10
1	1/0	2/–	–/–	3/–
2	4/–	2/1	5/–	–/–
3	1/–	–/–	5/–	3/0
4	4/–	2/–	–/–	6/–
5	–/–	2/–	5/–	6/–
6	1/–	–/–	5/–	6/1

10.12 Repita el problema 10.11 para la siguiente tabla de flujo primitiva:

	x_1x_2 00	01	11	10
1	1/0	2/–	–/–	4/–
2	1/–	2/0	3/–	–/–
3	–/–	2/–	3/0	8/–
4	5/–	–/–	7/–	4/1
5	5/1	6/–	–/–	4/–
6	5/–	6/1	7/–	–/–
7	–/–	6/–	7/1	8/–
8	1/–	–/–	3/–	8/0

10.13 Especifique una realización de circuito para la siguiente tabla de flujo reducida. Utilice la asignación de estados dada. Suponga que dispone de compuertas AND, OR y NOT para la realización.

y_1y_2	x_1x_2 00	01	11	10
00 *a*	*a*/0	*a*/1	*b*/–	*c*/–
01 *b*	*a*/–	*b*/0	*b*/0	*d*/–
11 *c*	*a*/–	*a*/–	*c*/1	*c*/1
10 *d*	*a*/–	*b*/–	*c*/–	*d*/0

10.14 Obtenga una tabla de flujo con un mínimo de filas compatible con la siguiente tabla de flujo primitiva:

	x_1x_2 00	01	11	10
1	1/1	6/–	–/–	5/–
2	2/0	4/–	–/–	3/–
3	2/–	–/–	9/–	3/0
4	2/–	4/0	7/–	–/–
5	1/–	–/–	7/–	5/1
6	1/–	6/1	7/–	–/–
7	–/–	4/–	7/0	10/–
8	8/0	4/–	–/–	10/–
9	–/–	6/–	9/1	3/–
10	1/–	–/–	9/–	10/0

10.15 Repita el problema 10.13 para la siguiente tabla de flujo, pero suponga que sólo dispone de compuertas NAND para el circuito.

y_1y_2		x_1x_2 00	01	11	10
00	*a*	*a*/00	*b*/–	*a*/00	*d*/–
01	*b*	*a*/–	*b*/01	*b*/01	*c*/–
11	*c*	*d*/–	*c*/10	*c*/10	*c*/01
10	*d*	*d*/00	*c*/–	*c*/–	*d*/10

10.16 Repita el problema 10.13 para la siguiente tabla de flujo reducida.

$y_1y_2y_3$		00	01	11	10
		x_1x_2			
000	a	a/1	c/–	b/–	a/1
001	b	b/0	d/–	b/0	a/–
010	c	a/–	c/0	e/–	c/0
101	d	b/–	d/1	f/–	d/1
110	e	e/1	f/–	e/1	c/–
100	f	a/–	f/0	f/1	a/–

10.17 Dada la siguiente tabla de excitación:

y_1y_2 \ x_1x_2	00	01	11	10
00	00	00	11	01
01	11	01	10	01
11	00	00	10	10
10	11	10	11	10

Y_1Y_2

(a) Localice todas las condiciones de competencia en la tabla.

(b) ¿Son o no son críticas las competencias?

(c) ¿Existen ciclos en la tabla?

10.18 Analice el circuito de la figura P10.18 y determine si el circuito tiene una competencia crítica. En tal caso, trace un diagrama de tiempos para mostrar el efecto que tiene la competencia sobre la respuesta del circuito.

10.19 Repita el problema 10.18 para el circuito de la figura P10.19.

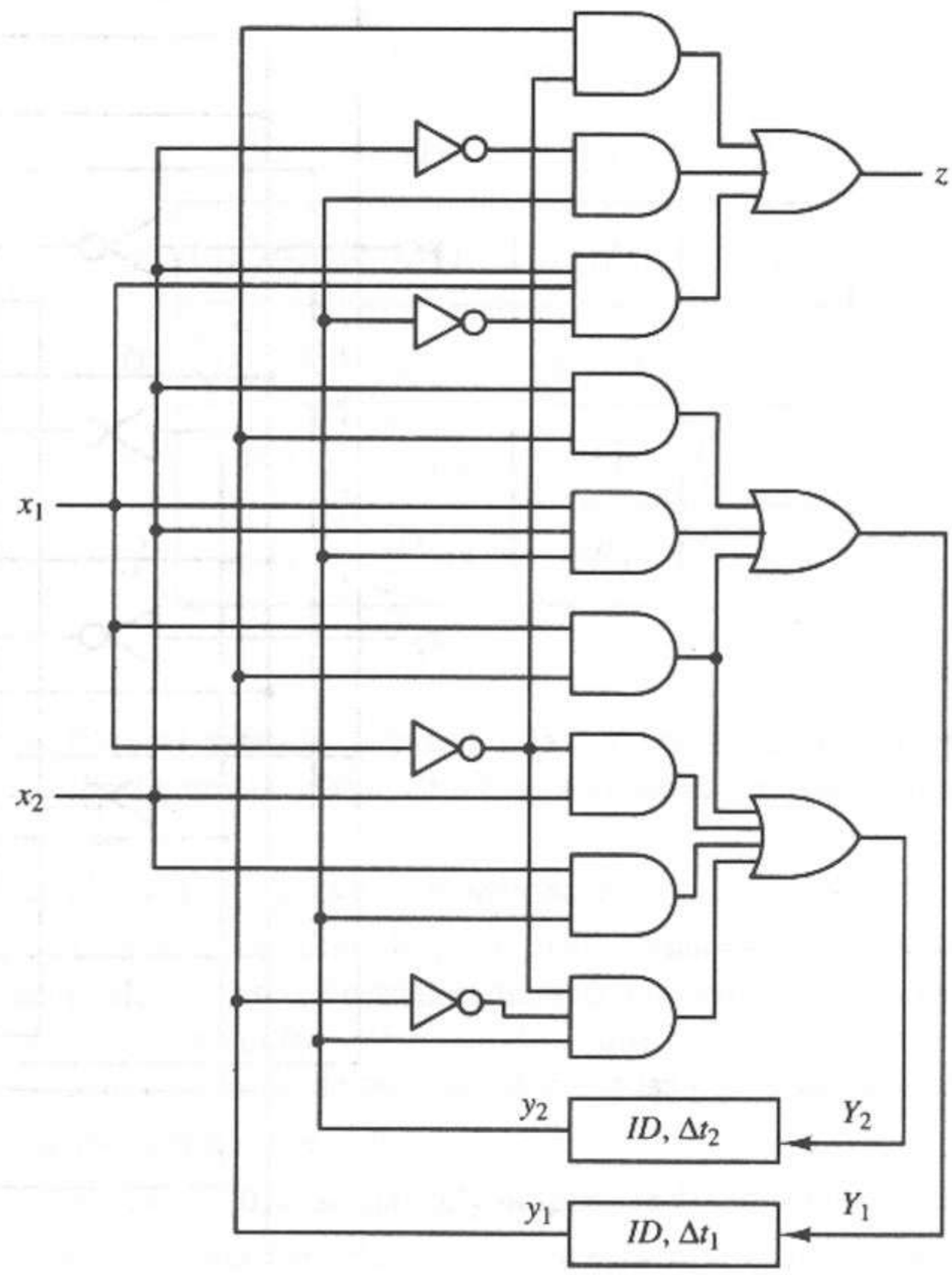


Figura P10.18

10.20 Obtenga una asignación de estados sin competencias críticas para la siguiente tabla de flujo reducida. Construya la tabla de excitación correspondiente.

	x = 0	x = 1
a	a/0	d/–
b	b/1	c/–
c	a/0	c/0
d	b/0	d/1

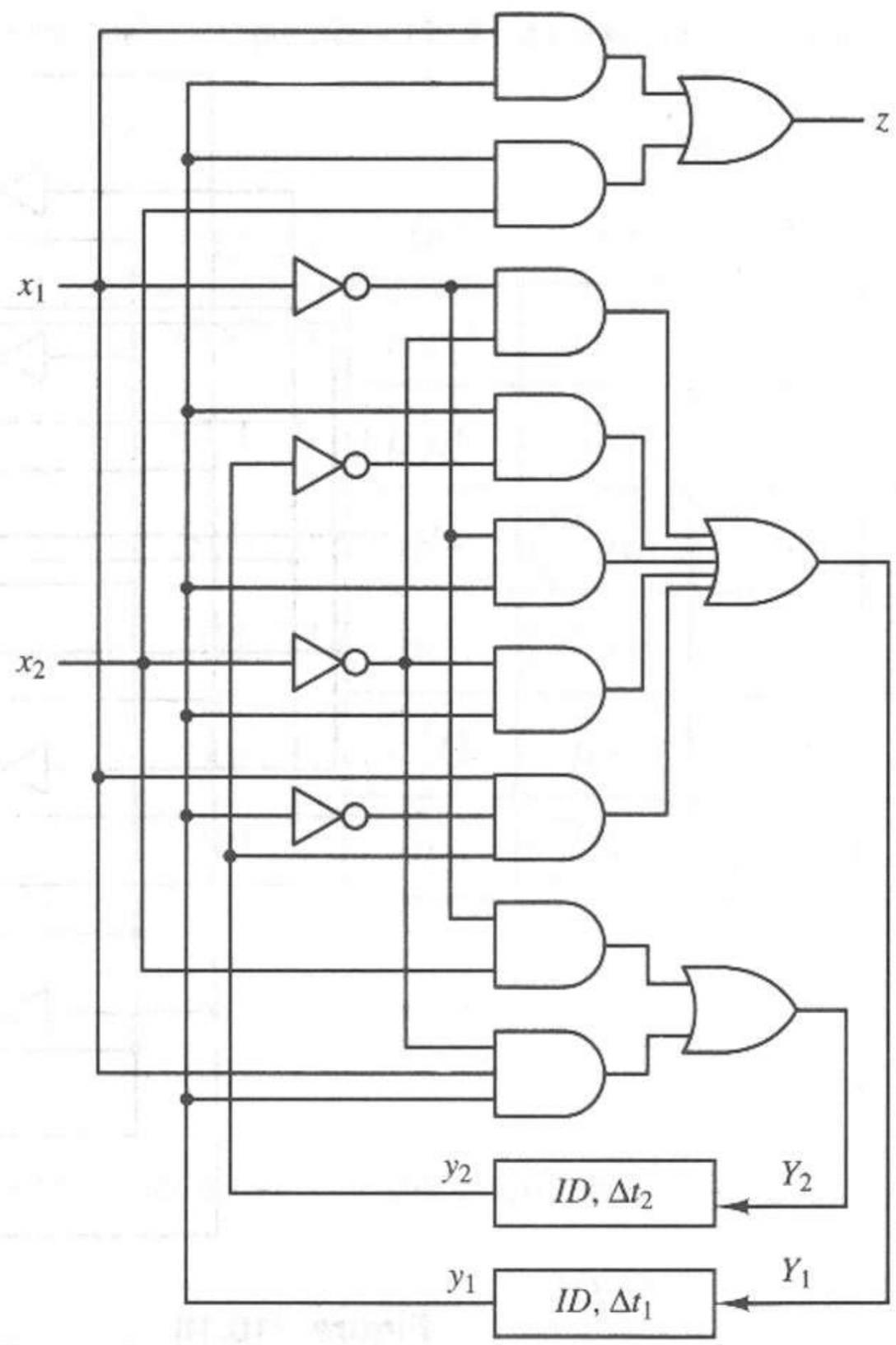


Figura P10.19

10.21 Repita el problema 10.20 para la siguiente tabla de flujo reducida:

	x_1x_2 00	01	11	10
a	a/0	b/–	c/–	a/0
b	b/1	b/1	b/1	c/1
c	a/–	b/1	c/1	c/1

10.22 Dada la siguiente tabla de flujo reducida:

	x_1x_2 00	01	11	10
a	$a/0$	$d/0$	$a/1$	$c/0$
b	$b/0$	$c/-$	$b/0$	$d/-$
c	$b/0$	$c/1$	$a/1$	$c/0$
d	$a/0$	$d/0$	$b/0$	$d/1$

(a) Utilice el método 1 y obtenga una asignación sin competencias críticas para la tabla. Construya la tabla de excitación correspondiente.

(b) Repita la parte (a) usando el método 2.

10.23 Hay que diseñar un circuito de modo fundamental para que funcione como una cerradura electrónica. La cerradura tiene dos interruptores de entrada (x_1 y x_2). Diseñe el circuito de modo que se produzca la señal de apertura ($z = 1$) sólo después de satisfacer las siguientes condiciones:

1. Se inicia con $x_1 = x_2 = 0$.
2. Mientras $x_2 = 0$, x_1 se activa, y después se desactiva dos veces.
3. Cuando x_1 está apagada, x_2 se activa para abrir la cerradura.

10.24 La figura P10.24 y las ecuaciones siguientes definen un circuito secuencial asíncrono de modo fundamental.

$$Y_1 = \bar{x}_2 y_2 + x_1 y_1 + x_1 \bar{x}_2$$
$$Y_2 = \bar{x}_1 y_2 + \bar{x}_1 x_2 + x_2 y_1$$
$$z = x_1 \bar{x}_2 + x_2 \bar{y}_1 + \bar{x}_1 y_2$$

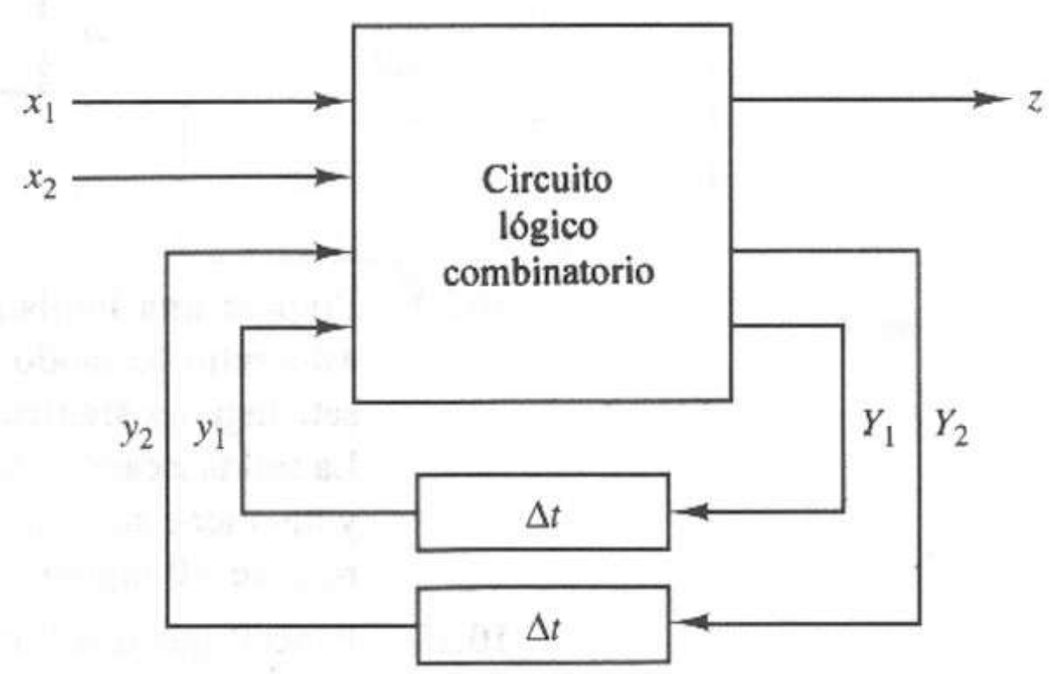


Figura P10.24

(a) Obtenga una tabla de flujo.

(b) Utilice la tabla de flujo preparada en la parte (a) para determinar la secuencia de salida correspondiente a la secuencia de entrada x_1x_2 = 00, 01, 11, 10, 11, 01, 00, 10 si las líneas de retardo se encuentran inicialmente en cero (estado estable: $x_1 = x_2 = y_1 = y_2 = 0$).

10.25 Dada la siguiente tabla de flujo reducida, determine una asignación de estados secundarios sin competencias críticas para este circuito secuencial asíncrono. Determine una realización NOR de dos niveles con elementos de retardo inercial.

	x_1x_2 00	01	11	10
a	a/0	b/–	a/1	b/–
b	a/–	b/0	c/–	b/0
c	a/–	c/1	c/0	b/–

10.26 Obtenga una realización NAND de dos niveles para la siguiente tabla de flujo primitiva.

	x_1x_2 00	01	11	10
a	a/0	b/–	–/–	c/–
b	a/–	b/1	d/–	–/–
c	a/–	–/–	d/–	c/1
d	–/–	b/–	d/0	e/–
e	a/–	–/–	d/–	e/0

10.27 Prepare una implantación NOR de dos niveles para un circuito secuencial asíncrono de modo fundamental con dos entradas (x_1,x_2) y una salida (z) que satisfaga las siguientes condiciones: Primero, z siempre es cero cuando $x_2 = 1$. La salida z cambia a 1 lógico en la primera transición $0 \rightarrow 1$ de x_1 cuando $x_2 = 0$ y mantiene su valor 1 lógico hasta que x_2 asume el valor 1 lógico y hace que z regrese a 0 lógico.

10.28 Especifique una implantación NAND de dos niveles para un circuito de modo fundamental con dos entradas (x_1,x_2) y una salida (z) que satisfaga las siguientes condiciones: Primero, $z = 0$ cuando $x_1 = 0$. La salida z cambia a 1 lógico en la primera transición $1 \rightarrow 0$ de x_2 cuando $x_1 = 1$. La salida mantiene el valor 1 lógico hasta que x_1 regresa a 0.

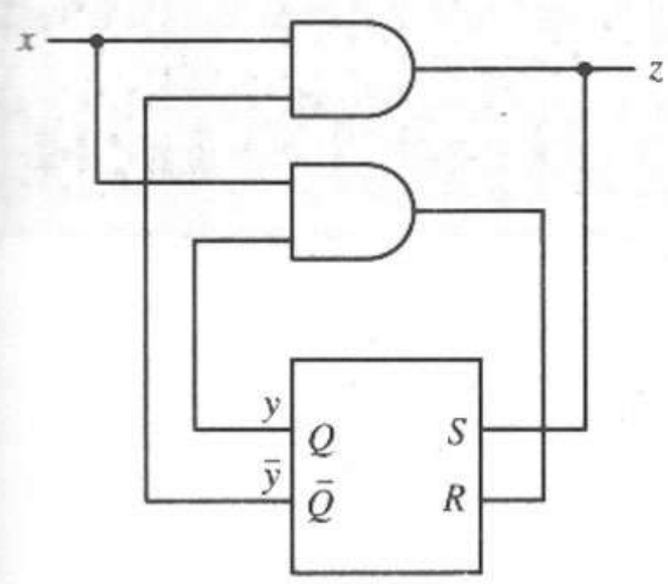


Figura P10.29

10.29 Considere el circuito secuencial asíncrono de la figura P10.29. Si la entrada del circuito son pulsos síncronos, determine lo siguiente:

(a) La tabla de estados si $A = 0$, $B = 1$.

(b) El diagrama de estados.

(c) El diagrama de tiempos para $x = 010011010$ y $y^0 = 0$.

Sugerencia: Puede usar mapas K.

10.30 Analice el circuito secuencial asíncrono de la figura P10.30. Este circuito tiene pulsos síncronos como entrada x. Construya lo siguiente:

(a) Un diagrama de tiempos para la secuencia de entrada $x = 01101000$ y $y^0 = 0$.

(b) Una tabla de estados.

(c) Un diagrama de estados.

En su solución, puede definir el ancho de los pulsos de la entrada x como igual al retardo del *flip-flop* T. Analice el efecto de la siguiente condición sobre el funcionamiento de este circuito secuencial: Permita que el pulso de entrada sea un poco mayor que el retardo del *flip-flop*. Muestre sus conclusiones en el diagrama de tiempos de la parte (a).

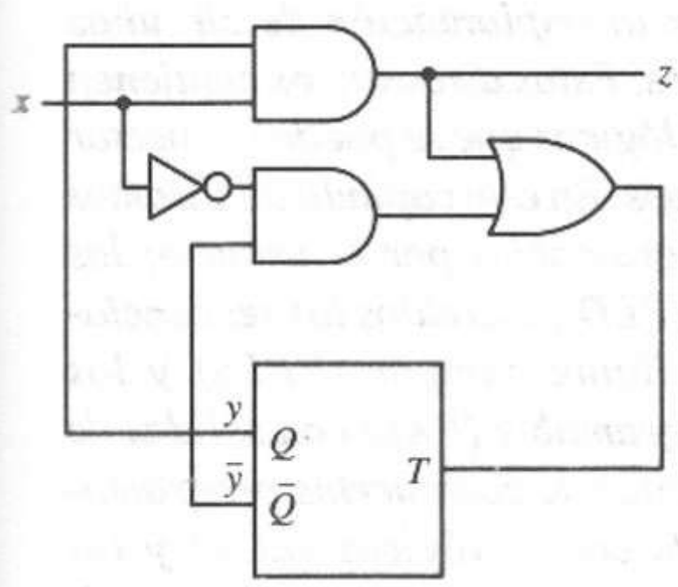


Figura P10.30

10.31 Analice el circuito secuencial asíncrono de la figura P10.31 si la entrada x del circuito tiene la forma de pulsos síncronos. Determine lo siguiente:

(a) El diagrama de tiempos si $x = 01010010100$ y $y_1^0 y_2^0 = 11$.

(b) La tabla de estados.

(c) El diagrama de estados.

Sugerencia. Los mapas K producen resultados incorrectos, pues se viola la premisa 3 para los circuitos de tipo pulso.

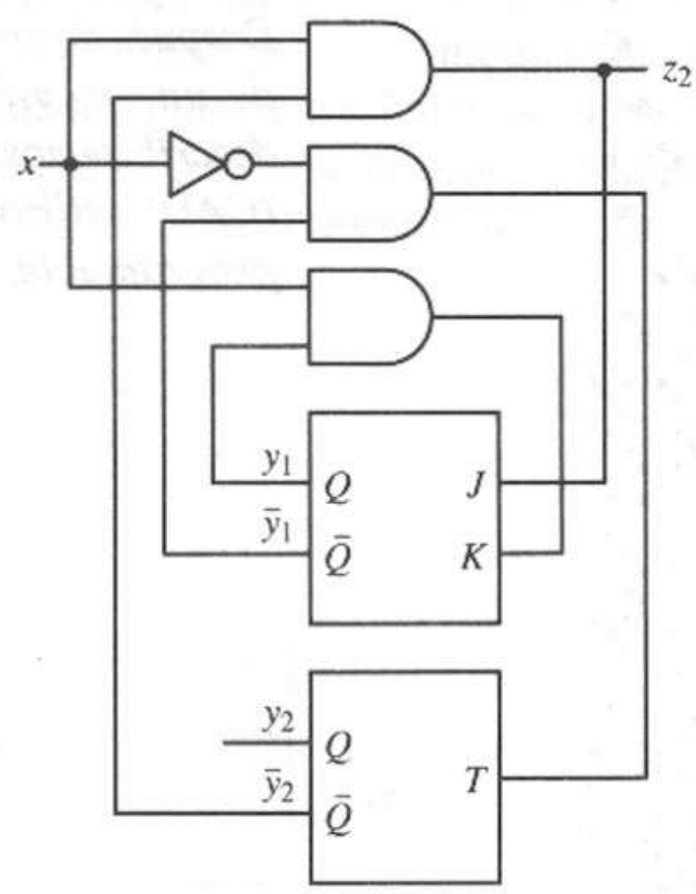
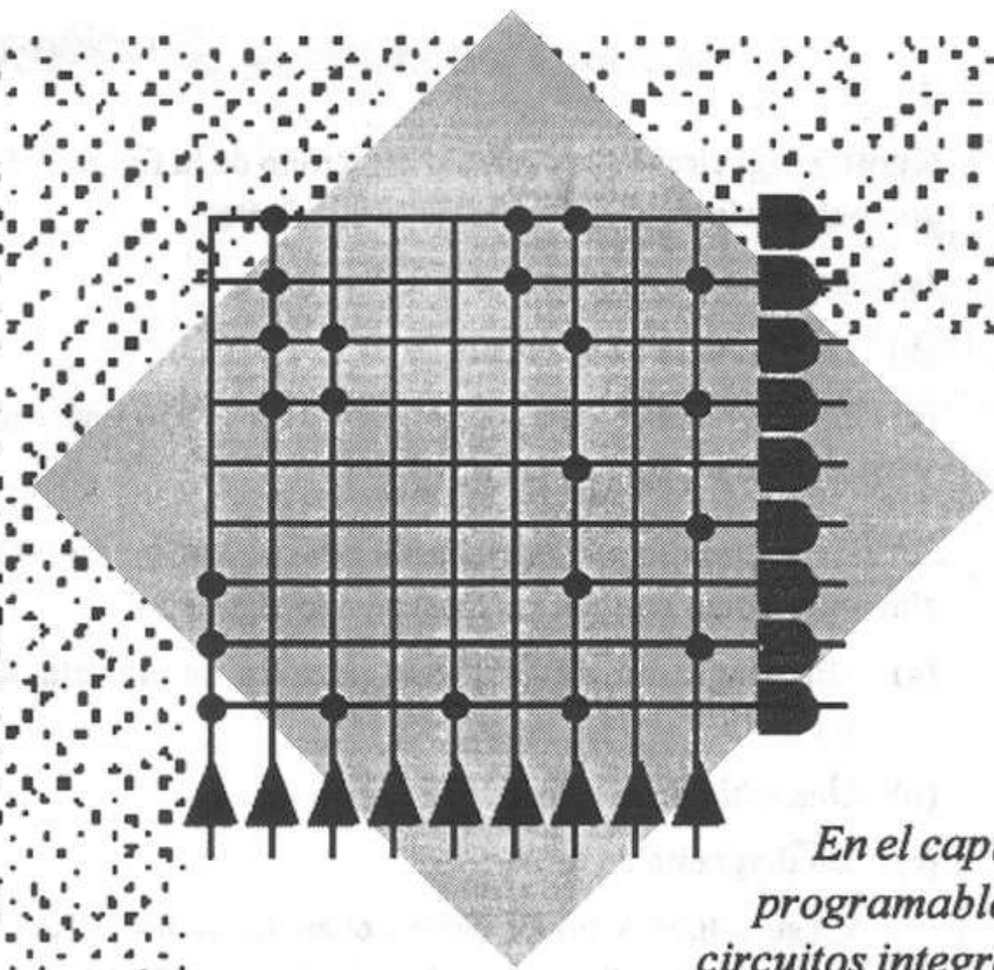


Figura P10.31

11



En el capítulo 5 analizamos los dispositivos lógicos programables (PLD), que se utilizan para reemplazar circuitos integrados a pequeña escala en la implantación de los circuitos lógicos combinatorios. Varios dispositivos programables apoyan también la implantación de circuitos secuenciales síncronos y asíncronos. Estos dispositivos contienen latches y/o flip-flops, o compuertas lógicas que se pueden conectar entre sí para formar latches o flip-flops. En este capítulo analizamos dos tipos básicos de circuitos programables por el usuario: los dispositivos lógicos programables (PLD), incluidos los secuenciadores lógicos programables mediante campos (FPLS) y los dispositivos lógicos de arreglo programable (PAL) con salidas de registros o macroceldas, y los arreglos de compuertas programables, que incluyen los arreglos de celdas lógicas (LCA) y los arreglos de compuertas programables mediante campos (FPGA). Después de analizar sus estructuras básicas, estudiaremos el diseño de un circuito secuencial con estos dispositivos. Por último, describiremos las herramientas de diseño asistido por computadora (CAD) utilizadas para crear circuitos secuenciales con lógica programable.

Circuitos secuenciales con dispositivos lógicos programables

11.1 Dispositivos lógicos programables con registros

Como vimos en el capítulo 8, el modelo básico de un circuito secuencial es el de la figura 11.1, con un bloque de lógica combinatoria y memoria. El estado del circuito se guarda en uno o más elementos de memoria, por lo general *flip-flops* o *latches*. Las entradas del bloque de lógica combinatoria son las entradas externas del circuito ($x_1,...,x_n$) y las variables de estado del circuito ($y_1,...,y_r$), que son las salidas de los elementos de memoria. El bloque de lógica combinatoria produce las salidas externas del circuito ($z_1,...,z_m$) y la información de estado siguiente ($Y_1,...,Y_r$) en forma de ecuaciones de excitación de *flip-flop*.

Con frecuencia se utilizan variantes del modelo básico de la figura 11.1 para ilustrar diversos aspectos de la estructura de un circuito secuencial. La más común consiste en dividir el bloque de lógica combinatoria para distinguir la generación de salidas de las variables de estado siguiente. Esto permite distinguir más fácilmente un modelo de Mealy de un modelo de Moore, definidos en el capítulo 8. La figura 11.2a muestra un modelo de Mealy, en el que las salidas y las variables de estado siguiente son funciones de las entradas y del estado actual. La figura 11.2b muestra un modelo de Moore, en el que las salidas sólo dependen de las variables de estado. En el caso de ciertos circuitos, como los contadores y

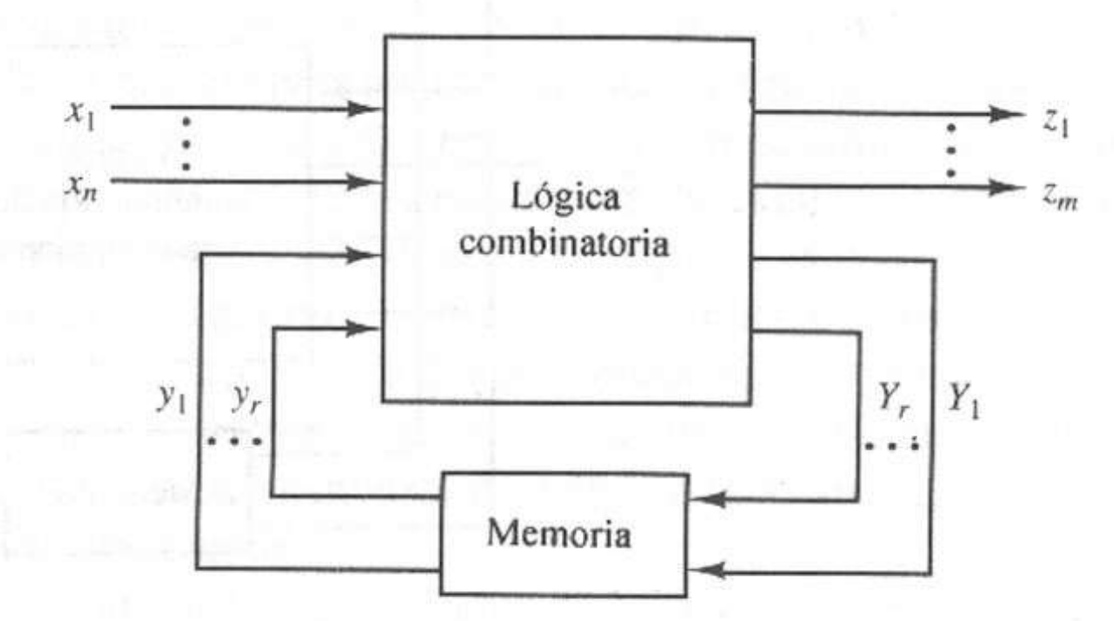


Figura 11.1 Modelo general de un circuito lógico secuencial.

los registros de corrimiento, las salidas del modelo de Moore son simplemente las variables de estado, como se muestra en la figura 11.2c.

En las figuras 11.2a y b las salidas del circuito son combinatorias; es decir, son salidas de circuitos lógicos combinatorios y, por tanto, son asíncronas respecto del reloj. Las salidas combinatorias responden de inmediato al cambio de las entradas o de las variables de estado. En muchos casos, es conveniente sincronizar los cambios de salida con un reloj, de la misma manera en que las variables de estado están sincronizadas con un reloj. Esto se logra agregando un registro de salida al circuito, como muestra la figura 11.3, de modo que el reloj controle

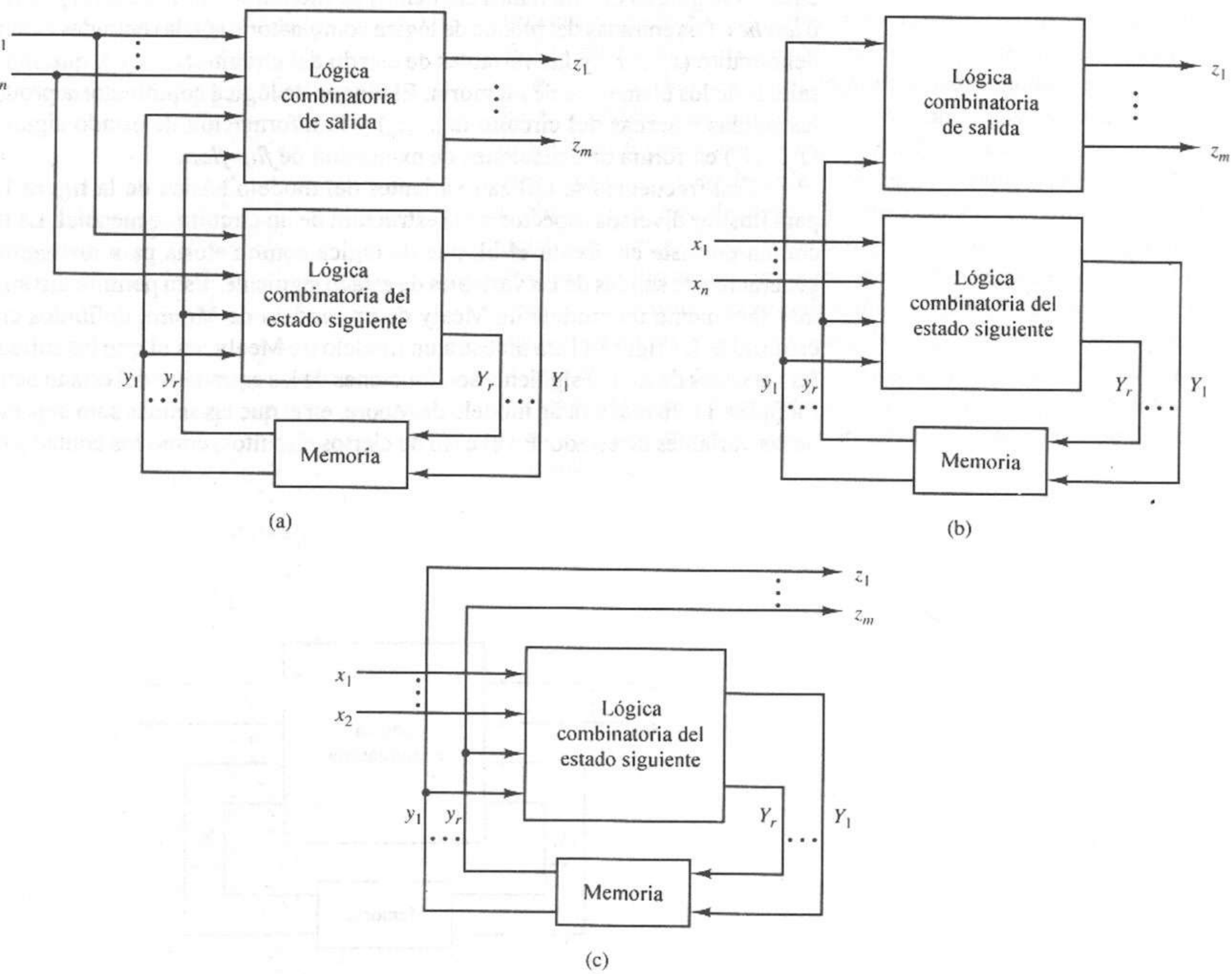


Figura 11.2 Modelos de un circuito secuencial con lógica combinatoria independiente para las salidas y el estado siguiente. **(a)** Modelo de Mealy. **(b)** Modelo de Moore. **(c)** Modelo de Moore (salidas = variables de estado).

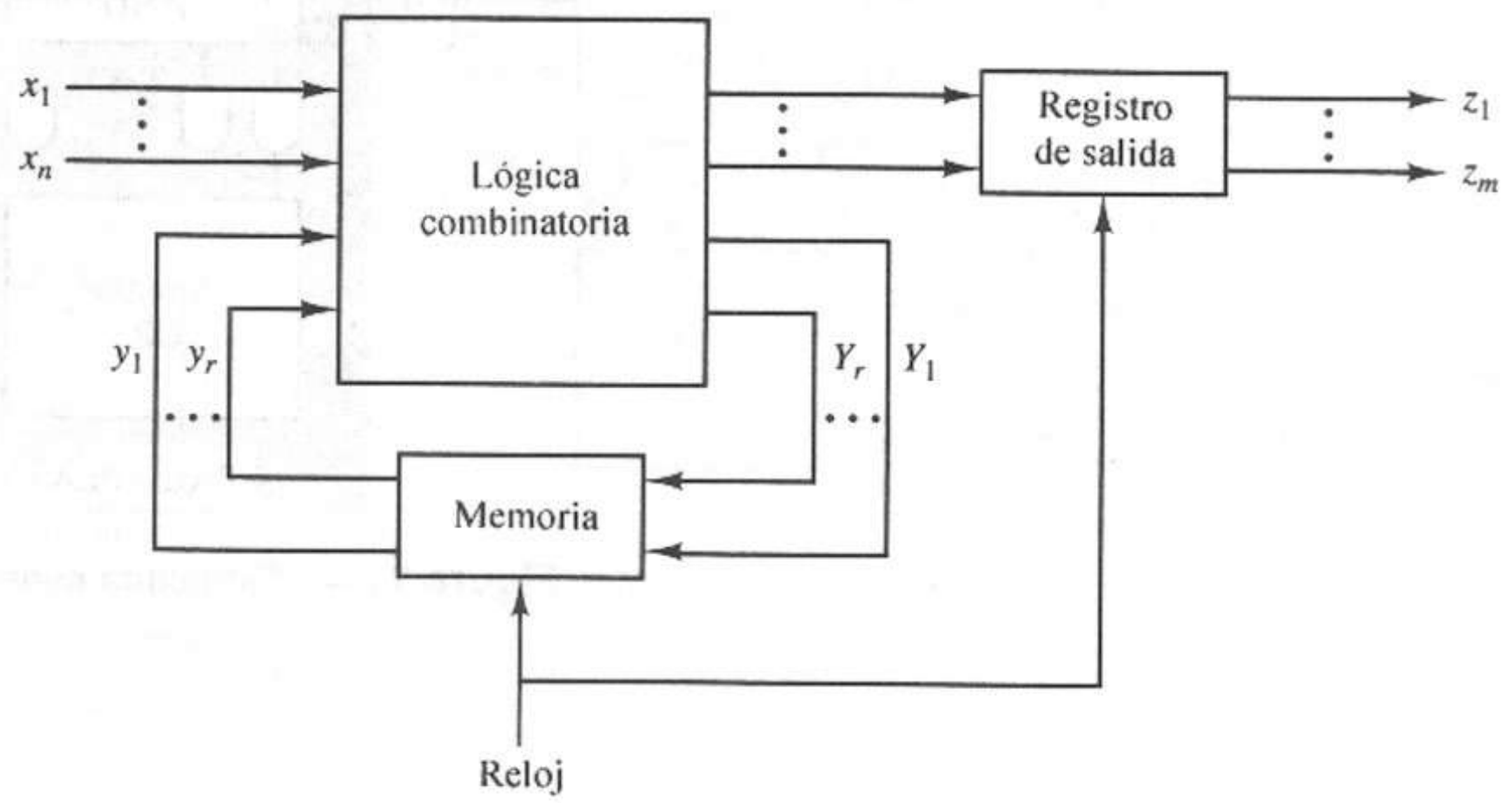


Figura 11.3 Modelo de circuito secuencial con salidas síncronas.

tanto el estado como los *flip-flops* de salida. Así, el registro de salida mantiene constantes las salidas hasta que el reloj dispara un cambio en el estado.

Los registros de salida se utilizan a menudo con circuitos lógicos no secuenciales, así como para mantener constantes las salidas de un circuito lógico combinatorio aunque las entradas cambien. Luego, un reloj hará que todas las salidas cambien en forma simultánea, una vez que las entradas están estables.

Los PLD descritos en el capítulo 5 son ideales para implantar los bloques lógicos combinatorios de un circuito secuencial, con la restricción de que todas las ecuaciones de excitación y de salida se expresen en forma SOP de dos niveles. Completamos el circuito secuencial agregando los elementos de memoria. Muchos PLD disponibles en el comercio contienen *flip-flops* o *latches* cuyas entradas de excitación están controladas por las salidas de un PLA o PAL estándar. Por lo general, las salidas de los *flip-flops* o *latches* se retroalimentan a las entradas de un circuito PLA/PAL; por tanto, estos dispositivos son ideales para implantar circuitos secuenciales de la forma de la figura 11.1.

La figura 11.4 muestra la estructura general de un PLD con registros. Como en el caso de los PLD combinatorios, un PLD con registros contiene un arreglo AND programable cuya salida alimenta un arreglo OR; el arreglo OR es programable en dispositivos basados en PLA y fijo en dispositivos basados en PAL. Cada salida del arreglo OR controla un *pin* de salida externo, en cuyo caso es una *salida combinatoria* o una entrada de excitación de un *flip-flop*. Las salidas de *flip-flop* se pueden conectar con *pines* externos, en cuyo caso éstos son *salidas con registros*, o bien las salidas de los *flip-flops* pueden retroalimentarse al arreglo AND sin conectarse con *pines* externos, en cuyo caso se les llama *registros ocultos*. Las salidas combinatorias y con registros también pueden retroalimentarse al arreglo AND. La figura 11.5 ilustra estas diferentes configuraciones.

Podemos implantar los modelos de Mealy de la forma de la figura 11.2a con PLD con registros como en la figura 11.6a, con salidas combinatorias para las funciones de salida y *flip-flops* para guardar las variables de estado. Como las variables de estado no tienen que ser visibles en las salidas, podemos utilizar registros ocultos o salidas con registros y retroalimentación. Los modelos de

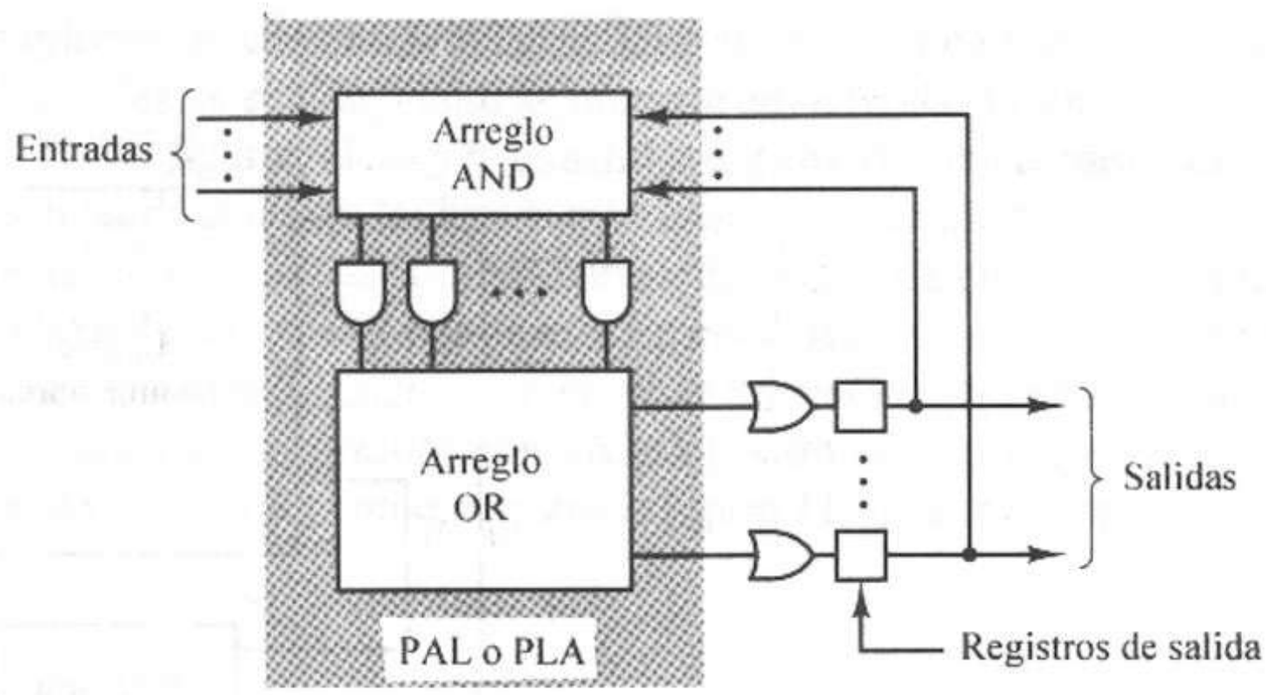


Figura 11.4 Estructura general de un PLD con registros.

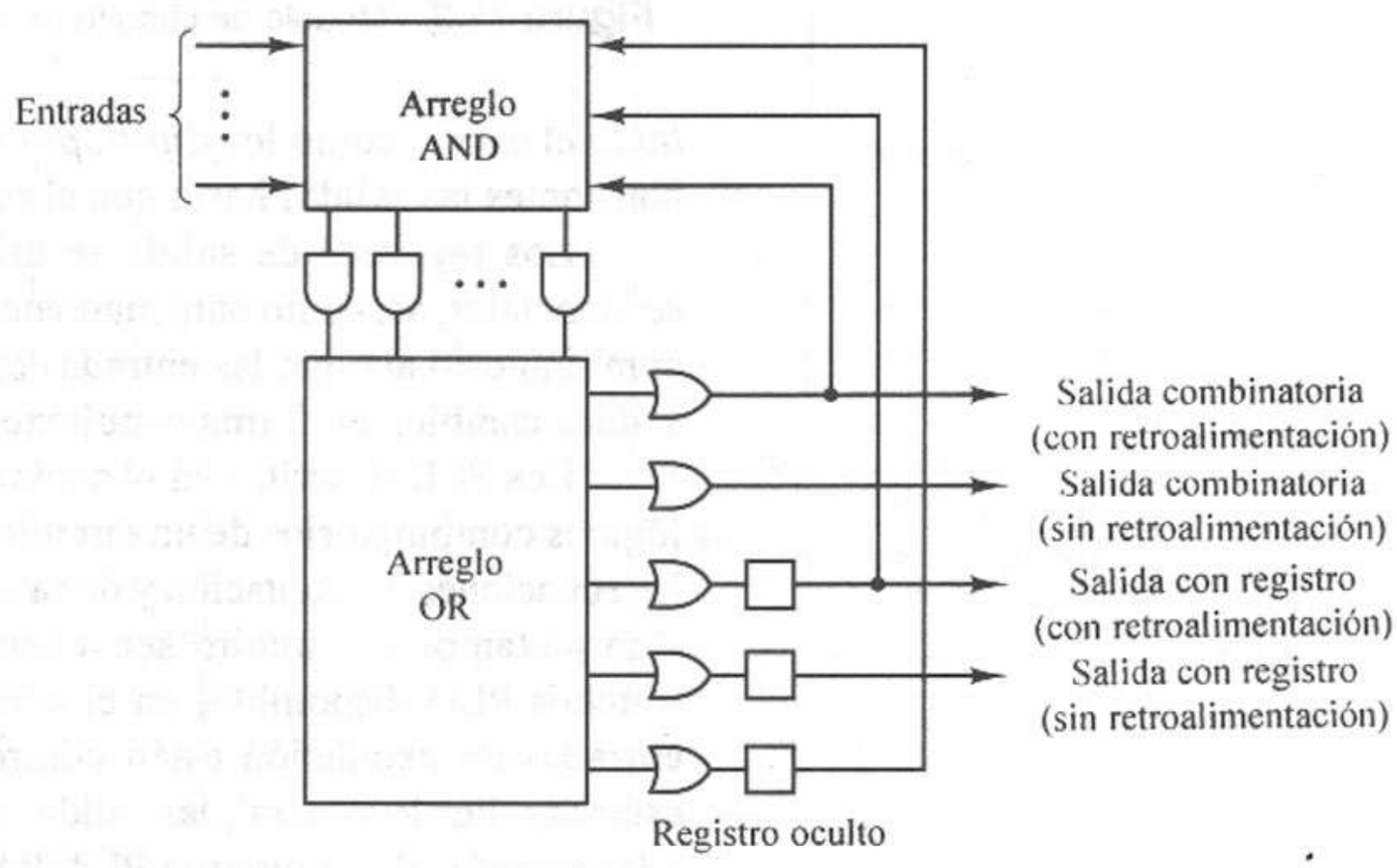


Figura 11.5 Opciones de salida de un PLD con registros.

Moore de la forma de la figura 11.2c requieren que los *flip-flops* de estado controlen las salidas externas, como en la figura 11.6b y, por tanto, requieren salidas con registros y retroalimentación. Implantamos los modelos de Moore de la forma de la figura 11.2b como en la figura 11.6a, donde las ecuaciones de salida se generan en el arreglo AND/OR sólo como funciones de las variables de estado.

En los modelos de Moore de la forma de la figura 11.2c no se necesitan salidas combinatorias. Sin embargo, todos los *flip-flops* de las variables de estado deben controlar salidas externas. Como ejemplos típicos están los contadores, los registros de corrimiento y los acumuladores.

Si queremos implantar una máquina secuencial con salidas síncronas como la de la figura 11.3, necesitamos salidas con registros para formar un *latch* con las salidas, aunque estos valores de salida no tengan que retroalimentarse al arreglo AND.

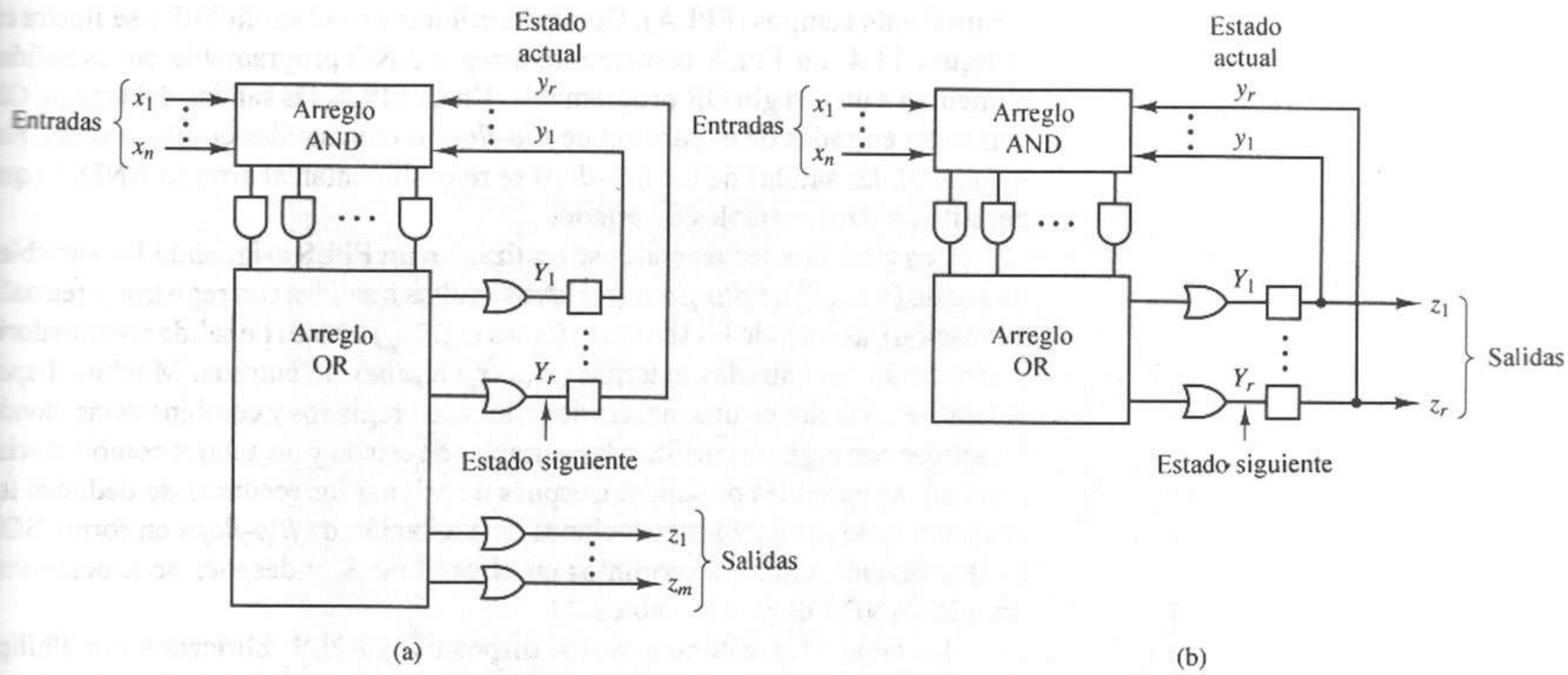


Figura 11.6 Modelos de un circuito secuencial asociado a PLD con registros. **(a)** Modelo de Mealy. **(b)** Modelo de Moore ($z_i = y_i$).

Con frecuencia, los PLD secuenciales incluyen varias otras características programables de utilidad. En algunos dispositivos, se puede programar el modo de operación del *flip-flop*, con lo que el diseñador puede configurar cada elemento de almacenamiento como un *flip-flop* D, T, JK o SR, según el diseño. Otra opción común es la polaridad de salida programable, lo que permite que las salidas de los *flip-flop* verdaderas (Q) o complementadas ($\bar{Q}$) controlen los *pines* de salida externa.

Muchos *flip-flops* PLD tienen entradas *clear* o *preset* síncronas o asíncronas. La mayor parte de las entradas *clear/preset* son controladas mediante términos producto simples generados en el arreglo AND, aunque algunos dispositivos conectan un *pin* de control externo a estas entradas para facilitar la implantación de una línea *reset* externa.

Algunos PLD apoyan el diseño de máquinas secuenciales asíncronas al permitir que las entradas de reloj de cada *flip-flop* individual sean controladas mediante términos del arreglo AND/OR, en vez de sincronizar todos los *flip-flops* con una única señal de reloj a partir de un *pin* externo al dispositivo. En algunos casos, el usuario puede elegir entre el funcionamiento síncrono o asíncrono para cada *flip-flop*.

En las siguientes secciones examinaremos dos clases principales de PLD utilizados en aplicaciones de máquinas secuenciales síncronas: los dispositivos basados en PLA, llamados secuenciadores lógicos programables mediante campos, y los dispositivos PAL con registros.

11.1.1 Secuenciadores lógicos programables mediante campos

El *secuenciador lógico programable mediante campos* (FPLS), introducido por Signetics en 1979 [1], es uno de los elementos lógicos programables más antiguos, desarrollado para apoyar la implantación de circuitos lógicos secuenciales. Un dispositivo FPLS típico está organizado en torno de un arreglo lógico programa-

ble mediante campos (FPLA). Como describimos en el capítulo 5 y se ilustra en la figura 11.4, un FPLA contiene un arreglo AND programable cuyas salidas alimentan a un arreglo OR programable. En un FPLS, las salidas del arreglo OR controlan entradas de excitación de *flip-flops* o bien salidas combinatorias. Por lo general, las salidas de los *flip-flops* se retroalimentan al arreglo AND, lo que permite realizar variables de estado.

Los circuitos secuenciales se realizan en un FPLS asignando las variables de estado ($y_1,...,y_r$) a *r flip-flops* (registros ocultos o salidas con registros y retroalimentación), asignando las salidas externas ($z_1,...,z_m$) a *pines* de salida combinatoria y asignando las entradas externas ($x_1,...,x_n$) a *pines* de entrada. Muchos dispositivos FPLS incluyen una mezcla de salidas con registros y combinatorias, donde las salidas con registros realizan las variables de estado y las salidas combinatorias realizan las variables de salida. Después de asignar los recursos, se deducen las ecuaciones de salida y las ecuaciones de excitación de *flip-flops* en forma SOP de dos niveles, como describimos en el capítulo 8, y después se asocian con arreglos AND/OR programables.

La tabla 11.1 enumera varios dispositivos FPLS fabricados por Philips [1], similares a los dispositivos disponibles de varios fabricantes. Examinaremos con detalle dos de ellos: el PLS105, que contiene *flip-flops* SR, y el PLS155, que contiene *flip-flops* configurables por el usuario.

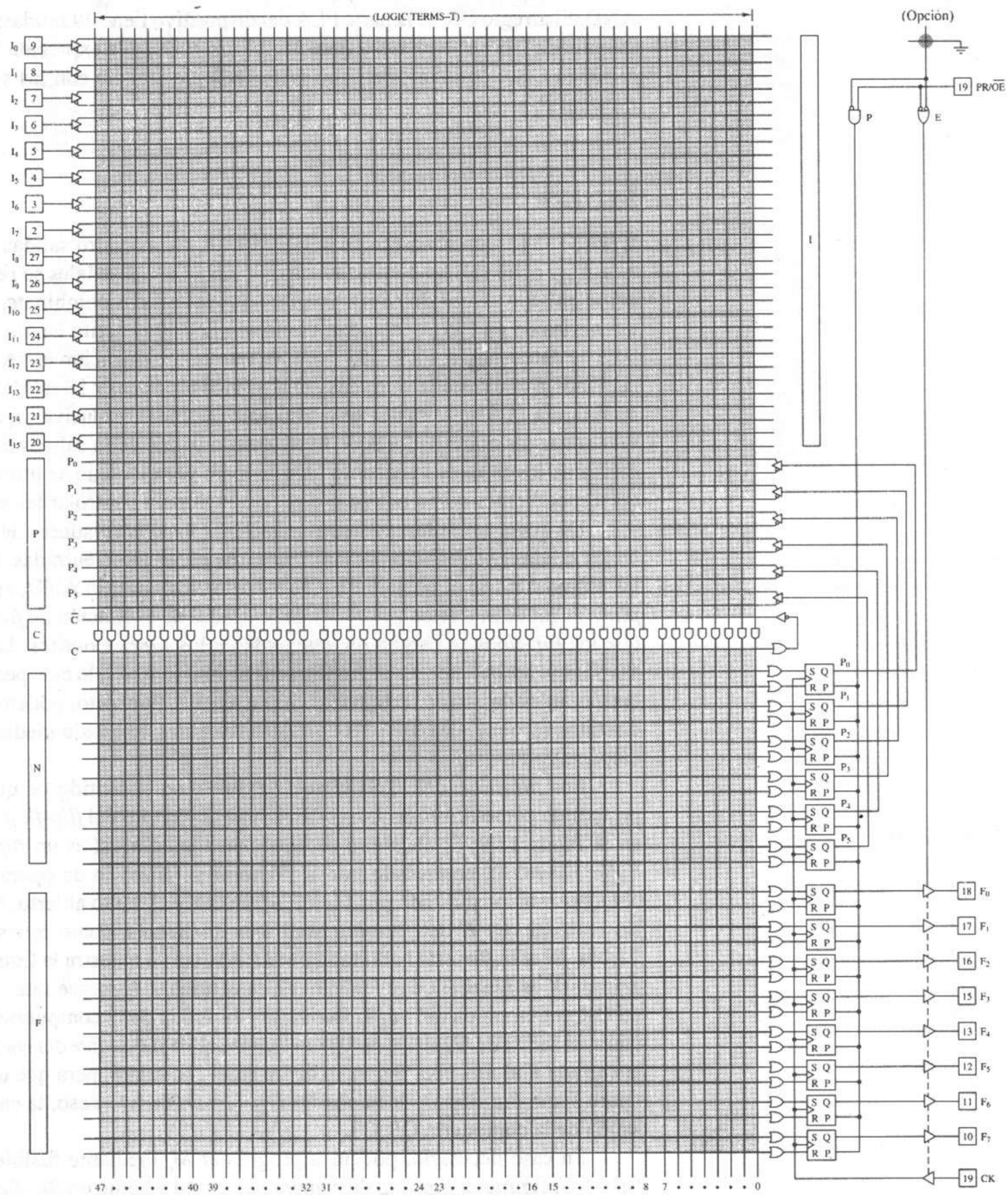
TABLA 11.1 DISPOSITIVOS FPLS DE SIGNETICS [1]

Dispositivo	Organización	Salidas con registro	Registros ocultos	Tipo de *flip-flop*
PLS105	16 × 48 × 8	8	6	SR
PLS155	16 × 45 × 12	4	0	D/JK/T
PLS157	16 × 45 × 12	6	0	D/JK/T
PLS167	14 × 48 × 6	6	6	SR
PLS168	12 × 48 × 8	8	6	SR
PLS179	20 × 45 × 12	8	6	D/JK/T

PLS105

El dispositivo FPLS PLS105 [1] fue uno de los primeros dispositivos FPLS disponibles en forma comercial y se basa en un dispositivo FPLA PLS100. Como muestra la figura 11.7, el PLS105 tiene 14 *flip-flops* SR. Las salidas de ocho de los *flip-flops* se conectan a *pines* de salida externa y no se retroalimentan al arreglo AND, de modo que no son adecuados para implantar variables de estado. Estos *flip-flops* pueden servir para crear un registro de salida. Las salidas de los otros seis *flip-flops*, P_5 a P_0, se retroalimentan al arreglo AND, pero no están conectados a *pines* de salida externa. En esta configuración, los últimos *flip-flops* son *registros ocultos*, ya que están contenidos dentro del circuito sin que exista un acceso directo a sus salidas. Así, podemos realizar máquinas secuenciales de hasta seis variables de estado y ocho salidas con un solo PLS105.

Los 14 *flip-flops* del PLS105 se controlan mediante un *pin* de entrada de reloj común y un *pin* de entrada *preset* común, PR/$\overline{\text{OE}}$. Si no se necesita la



NOTAS:
1. Todas las entradas a compuertas AND con una conexión eliminada flotan a "1" lógico.
2. Todas las entradas a compuertas OR con una conexión eliminada flotan a "0" lógico.
3. ● Conexión programable.

Figura 11.7 FPLS PLS105 de Philips [1]. *Fuente*: Philips, "Programmable Logic Devices (PLD) Data Handbook", Philips Semiconductors, Sunnyvale, CA, 1994.

opción *preset*, es posible utilizar el *pin* PR/$\overline{OE}$ para habilitar los controladores de triple estado conectados al registro de salida.

La organización global del PLS105 es de 16×48×8, con 16 entradas exclusivas I_{15} a I_0, 8 salidas F_7 a F_0 y 48 términos producto generados en el arreglo AND. El arreglo OR de la parte PLA del dispositivo tiene 29 salidas; 28 controlan las entradas de excitación S y R de los 14 *flip-flops*, y la expresión SOP complementada C se retroalimenta al arreglo AND juntamente con las seis salidas de *flip-flop* P_5 a P_0.

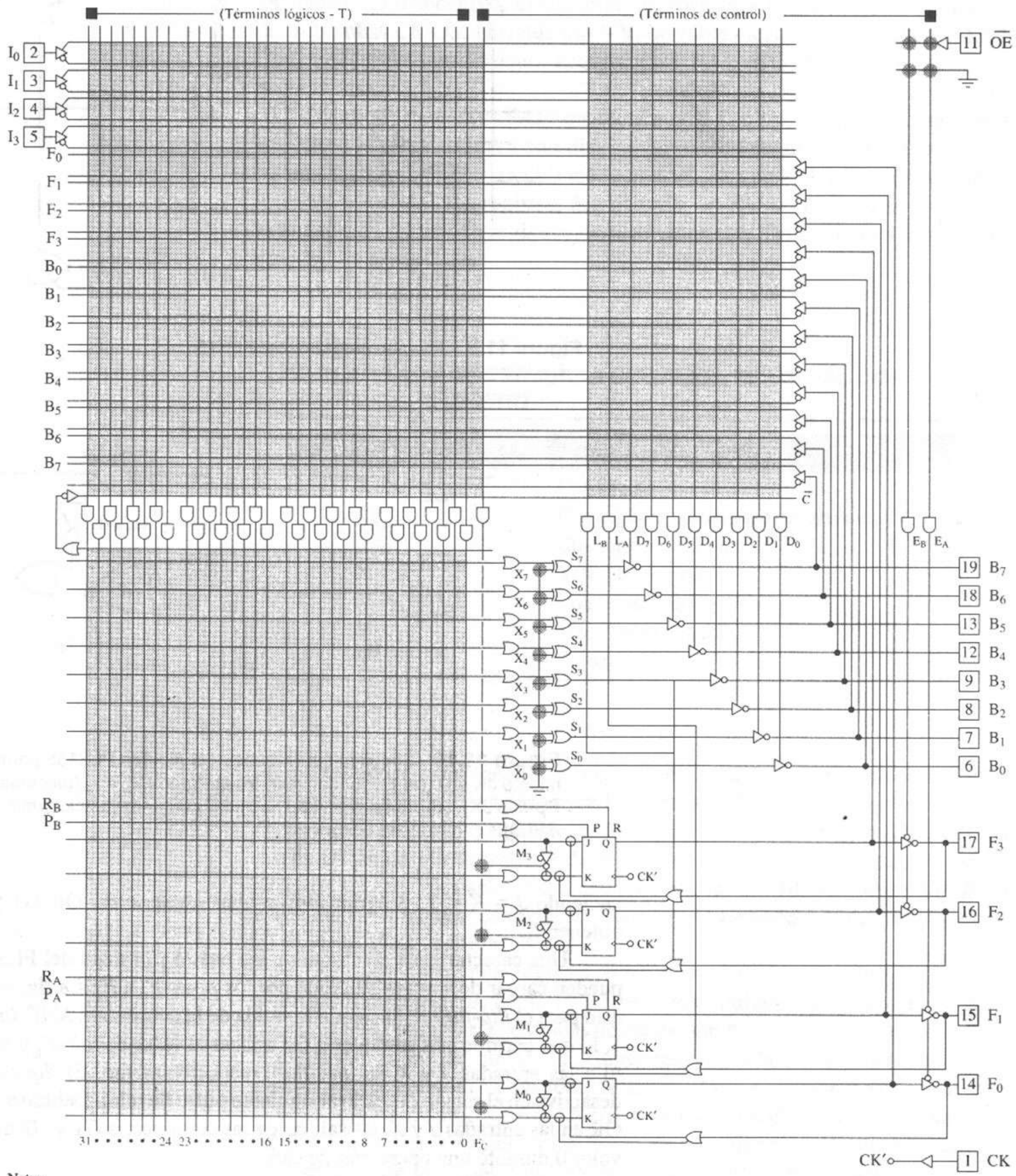
PLS155

El PLS155 [1], que aparece en la figura 11.8, tiene cuatro salidas con registro, F_3 a F_0, y ocho salidas combinatorias, B_7 a B_0. Las 12 señales se retroalimentan como entrada al arreglo AND. Observe que las salidas combinatorias son similares a las de un FPLA estándar, como el PLS100 descrito en el capítulo 5. La hoja de datos para el PLS155 [1] indica que su organización es 16 × 45 × 12, lo cual indica que tiene 16 entradas, 45 términos producto y 12 salidas. Un análisis de la figura 11.8 muestra que sólo hay cuatro entradas exclusivas, I_3 a I_0. Podemos inhabilitar los controladores de triple estado de las ocho salidas combinatorias mediante los términos producto D_7 a D_0 para permitir que se utilicen los *pines* correspondientes como entradas o se habiliten para controlar las salidas.

De manera similar, podemos inhabilitar los controladores de triple estado de las salidas con registro para utilizar estos *pines* como entradas. En este caso, las entradas no se alimentan de manera directa al arreglo AND, sino que estos *pines* introducen ciertos valores en las entradas de excitación de *flip-flop*, por lo que los *flip-flops* se cargan directamente de los *pines* externos. Las salidas de *flip-flop* se retroalimentan como entradas al arreglo AND, lo cual permite guardar las variables de estado de un circuito secuencial. Por tanto, podemos implantar circuitos secuenciales con más de cuatro variables de estado mediante un único dispositivo PLS155.

Los *flip-flops* del PLS155 son flexibles, en el sentido de que se pueden programar como *flip-flops* JK o D. Examine el circuito del *flip-flop* F_0, ampliado en la figura 11.9. El elemento de almacenamiento en sí es un *flip-flop* JK. El *buffer derivado*, controlado por M_0, determina el modo de operación real. Si el *buffer* está inhabilitado por $M_0 = 1$, actúa como circuito abierto, haciendo que las entradas J y K del *flip-flop* sean independientes, como muestra la figura 11.10a. Si el *buffer* está habilitado por $M_0 = 0$, como muestra la figura 11.10b, la entrada K es igual al complemento de la entrada J. Observe que establecemos un OR con la salida del *buffer* derivado y la salida de la compuerta OR $G2$. Por tanto, la salida de $G2$ debe ser 0 para que la salida del *buffer* derivado determine la entrada K al *flip-flop*. Según lo dicho en el capítulo 6, para que un *flip-flop* K opere como *flip-flop* D, hacemos $J = D$ y $K = \overline{D}$. En este caso, la entrada D es la salida de la compuerta OR $G1$.

En caso necesario, podemos programar M_0 mediante fusibles como 0, a fin de configurar el *flip-flop* de manera permanente como un *flip-flop* D. De otro modo, el comportamiento del *flip-flop* queda determinado por la salida de la compuerta AND F_C, lo que permite al *flip-flop* cambiar de manera dinámica entre los modos de operación JK y D. Si se desea un *flip-flop* SR, basta utilizar la configuración del *flip-flop* JK con $J = S$ y $K = R$, sin permitir la combinación $S = R = 1$. De manera similar, podemos crear el equivalente de un *flip-flop* T



Notas:
1. Todas las entradas a compuertas OR con una conexión eliminada flotan a "0" lógico.
2. Las demás compuertas y entradas de control con una conexión eliminada flotan a "1" lógico.
3. ⊕ denota una conexión OR
4. ● Conexión programable.

Figura 11.8 FPLS PLS155 de Philips [1]. *Fuente*: Philips, "Programmable Logic Devices (PLD) Data Handbook", Philips Semiconductors, Sunnyvale, CA, 1994.

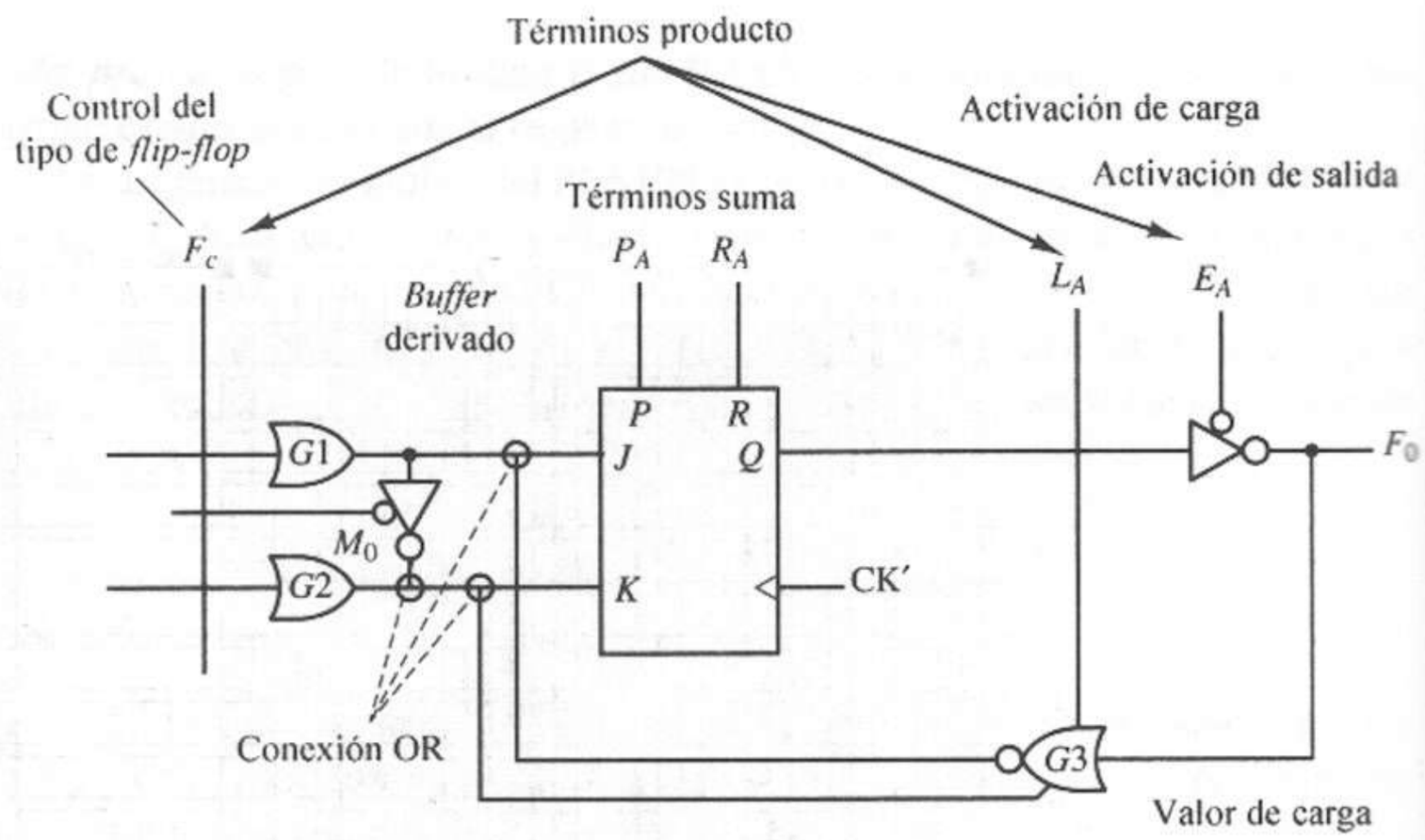


Figura 11.9 *Flip-flop* programable PLS155.

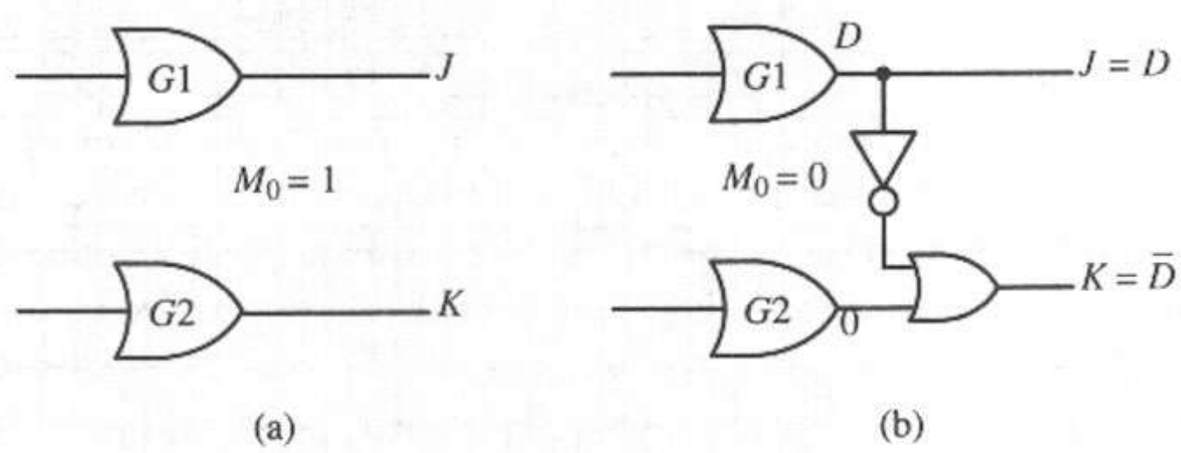


Figura 11.10 Circuitos equivalentes del *flip-flop* PLS155 para funcionamiento JK y D. **(a)** *Buffer* derivado inhabilitado ($M_0 = 1$) (funcionamiento de *flip-flop* JK). **(b)** *Buffer* derivado habilitado ($M_0 = 0$) (funcionamiento de *flip-flop* D).

haciendo $J = K = T$, programando ambas compuertas OR $G1$ y $G2$ para así obtener T.

Una característica adicional de los cuatro *flip-flops* del PLS155 es que se pueden cargar de manera síncrona por los *pines* F_3 a F_0. La figura 11.9 muestra esto para el *flip-flop* F_0 haciendo la salida de la compuerta AND *load* (L_A) igual a 1, lo que permite a la compuerta $G3$ aplicar el valor del *pin* F_0 y su complemento a las entradas J y K del *flip-flop*, respectivamente. El *flip-flop* se activa o desactiva en el estado deseado al siguiente pulso de reloj. Debido a las conexiones OR en las entradas J y K, las salidas de las compuertas $G1$ y $G2$ deben asumir el valor 0 durante una operación de carga.

11.1.2 PAL con registro

Los PAL con registro son similares a los dispositivos FPLS examinados antes, ya que contienen configuraciones PAL estándar con una o más salidas que controlan las entradas de excitación de *flip-flops*. La notación utilizada para

designar los PAL con registro es la misma que para PAL combinatorios: PAL*xyz*, donde *x* es el número total de entradas, *z* es el número total de salidas y *y* indica la arquitectura de salida. Con frecuencia se agregan otros sufijos para indicar la velocidad del dispositivo, el consumo de energía y la tecnología del circuito (TTL bipolar, CMOS, borrable por UV, borrable eléctricamente, etcétera) [2]. Con frecuencia, una misma arquitectura PAL está disponible en varias tecnologías.

Muchos dispositivos PAL son miembros de familias que utilizan una misma estructura básica PAL pero con diversas arquitecturas de salida. La tabla 11.2 resume las arquitecturas de salida más comunes. Para un PAL con registro simple, R indica las salidas con registro. Por ejemplo, un PAL16R6 tiene un total de 16 entradas y 6 salidas con registro. Analizaremos este dispositivo en la siguiente sección.

TABLA 11.2 ARQUITECTURAS DE SALIDA DE UN PAL CON REGISTRO

Código	Significado
R	Con registro
RP	Con registro y polaridad programable
RA	Con registro asíncrono
X	XOR con registro
A	Con registro aritmético
V	Macrocelda versátil
S	Secuenciador

Las salidas *asíncronas con registros* difieren de las salidas con registros estándar por el hecho de que las entradas de reloj de los *flip-flops* son controladas por términos producto o expresiones SOP generadas en el arreglo AND/OR, en vez de un único *pin* de entrada de reloj. Esto permite la implantación de los circuitos secuenciales asíncronos como en el capítulo 10, ya que los *flip-flops* individuales se pueden controlar de manera independiente como funciones de las entradas y las variables de estado.

Los *dispositivos con registro XOR*, como PAL16X4, y los dispositivos con registro *aritméticos*, como PAL16A4, son similares, pues las compuertas OR exclusivo controlan las entradas de excitación de los *flip-flops* D, facilitando la implantación de las funciones aritméticas, como la suma y la resta. Por lo general, las dos entradas a estas compuertas XOR son expresiones SOP generadas en el arreglo AND/OR del PAL. La arquitectura con registro aritmético también incluye una lógica especial en las líneas de retroalimentación para facilitar la implantación de las señales *carry/borrow* (acarreo/préstamo) entre los bits de diversos circuitos aritméticos.

La designación V de la arquitectura de salida indica la presencia de macroceldas lógicas *versátiles* en las salidas, que tienen varias opciones programables. Un ejemplo común es el PAL22V10, con 22 entradas y 10 salidas; cada una de éstas últimas tiene una macrocelda programable. Analizaremos el PAL22V10 con mayor detalle más adelante.

La tabla 11.3 enumera varios PAL con registros de uso común y sus configuraciones. El lector puede consultar [2] si desea más detalles acerca de éstos y otros dispositivos PAL.

TABLA 11.3 PAL CON REGISTROS [2]

Dispositivo	Entradas exclusivas	Salidas con registro exclusivas	*Pines* de E/S	Productos
PAL16R8	8	8	0	8
PAL16R6	8	6	2	8
PAL16R4	8	4	4	8
PAL16RP8	8	8	0	8
PAL16RA8	8	0–8	8–0	4
PAL16X4	8	4	4	8
PAL20R8	12	8	0	8
PAL20XRP8	10	8	2	8
PAL22V10	12	0–10	10–0	8–16
PAL32VX10	12	0–10	10–0	8–16
PAL23S8	9	4	4	8–12

PAL16R6

El PAL16R6 forma parte de la familia PAL16R8 de dispositivos PAL con registro de 20 *pines*. Como muestra la figura 11.11, el PAL16R6 tiene ocho *pines* de entrada exclusivos, I_8 a I_1, y ocho *pines* de salida, de los cuales dos son combinatorios (I/O_8 y I/O_1) y seis tienen registros (O_7 a O_2). Cada *pin* de salida con registro es controlado por un *flip-flop* D cuyas salidas y complementos se retroalimentan al arreglo AND. Las salidas combinatorias también se retroalimentan, para un total de 32 entradas al arreglo AND. Cada entrada de excitación del *flip-flop* D es controlada por una de las ocho salidas SOP del PAL. Por tanto, el PAL tiene un total de 32 entradas y 8 salidas. Los seis *flip-flops* son controlados por un único *pin* de entrada de reloj CLK. Los ocho *pines* de salida son controlados mediante controladores de triple estado, que a su vez son manejados por el único *pin* de control $\overline{OE}$.

Otros PAL con registros de la familia PAL16R8 son el 16R4 y 16R8, que utilizan el mismo circuito PAL básico, pero con diversas distribuciones de los *flip-flops* y las salidas combinatorias en los ocho *pines* de salida. Por ejemplo, el 16R4 tiene cuatro salidas combinatorias y cuatro salidas con registro, y el 16R8 tiene ocho salidas con registro, sin salidas combinatorias.

Otros dispositivos de la familia PAL16R8, como PAL16A4, PAL16X4 y PAL16RP8, utilizan el mismo circuito básico PAL, pero con diferentes arquitecturas de salida. Se han construido otras familias PAL con registros en torno de diferentes configuraciones PAL. Éstas se describen con detalle en [2].

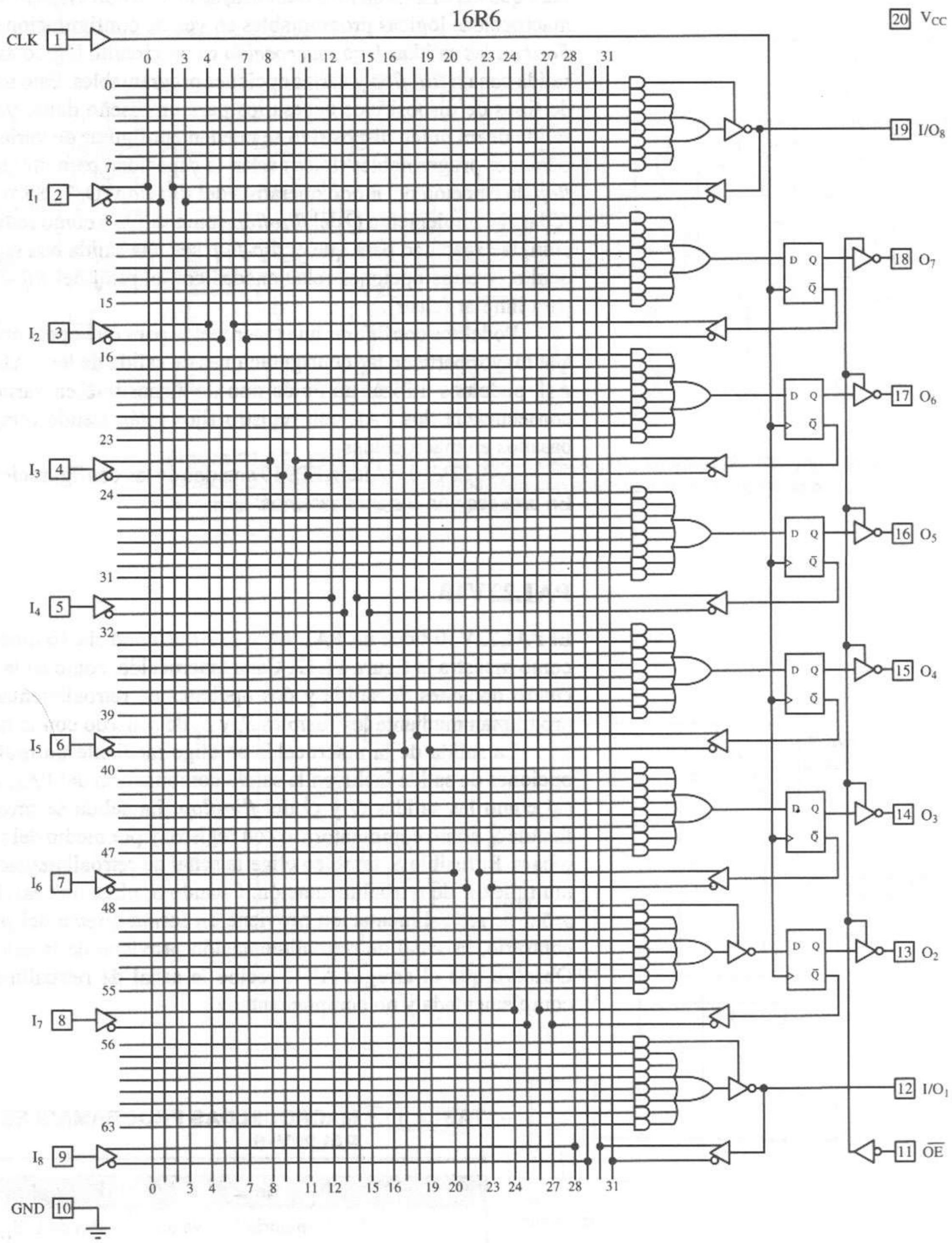


Figura 11.11 PAL16R6 de AMD [2]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

11.1.3 PLD con macroceldas lógicas programables

Para que los PLD sean más flexibles, se han creado dispositivos universales con macroceldas lógicas programables en vez de configuraciones sencillas de *flip-flops* en las salidas. Una *macrocelda* es un circuito lógico asociado a un *pin* de salida con un *flip-flop* y varias opciones programables. Esto minimiza la cantidad de tipos de dispositivos necesarios para un diseño dado, ya que las celdas de salida de un único dispositivo se pueden configurar de varias formas. Entre las opciones programables típicas están la capacidad para utilizar o ignorar el *flip-flop*, la elección del modo operativo del *flip-flop* (D, T, SR o JK), la elección de *(Q)* o el complemento $\overline{Q}$ del *flip-flop* como salida o como señal de retroalimentación, la capacidad para que el *flip-flop* sea una salida con registro o un registro oculto, y otras opciones relacionadas con el reloj del *flip-flop*, y las entradas para limpiar (clear).

Podemos configurar una macrocelda para que opere en forma equivalente a la mayor parte de las configuraciones de salida de los PAL con registro fijos. Así, podemos utilizar un único tipo de dispositivo en varias aplicaciones. En consecuencia, los PAL con registro fijos están siendo remplazados por PAL basados en macroceldas.

PAL22V10 y Altera EP910 son dos de las configuraciones más utilizadas. En esta sección las analizaremos.

PAL22V10

El PAL22V10 tiene un PAL 44 × 132 que controla 10 macroceldas de salida, como muestra la figura 11.12. Cada macrocelda, como en la figura 11.13, tiene cuatro opciones de salida y dos opciones de retroalimentación, las cuales se programan mediante los fusibles S_1 y S_0 de acuerdo con la tabla 11.4.

La salida de la macrocelda se elige mediante un multiplexor 4 a 1; las opciones de salida incluyen la salida combinatoria del PAL y su complemento, así como las salidas Q y $\overline{Q}$ del *flip-flop*. La salida se programa mediante el fusible S_1 como combinatoria o con registro y por medio del fusible S_0 como alta o baja. El fusible S_1 también elige la señal de retroalimentación controlando el multiplexor de retroalimentación. Cuando S_1 elige una salida combinatoria, la señal de retroalimentación proviene en forma directa del *pin* de E/S; en caso contrario, la señal de retroalimentación proviene de la salida $\overline{Q}$ del *flip-flop*. Observe que el arreglo AND recibe la señal de retroalimentación en forma complementada y no complementada.

TABLA 11.4 MACROCELDAS PROGRAMABLES DEL PAL22V10

S_1	S_2	Salida	Retroalimentación
0	0	Combinatoria (activa alta)	*Pin* de E/S
0	1	Combinatoria (activa baja)	*Pin* de E/S
1	0	Con registro (Q)	$\overline{Q}$ salida de *flip-flop*
1	1	Con registro ($\overline{Q}$)	$\overline{Q}$ salida de *flip-flop*

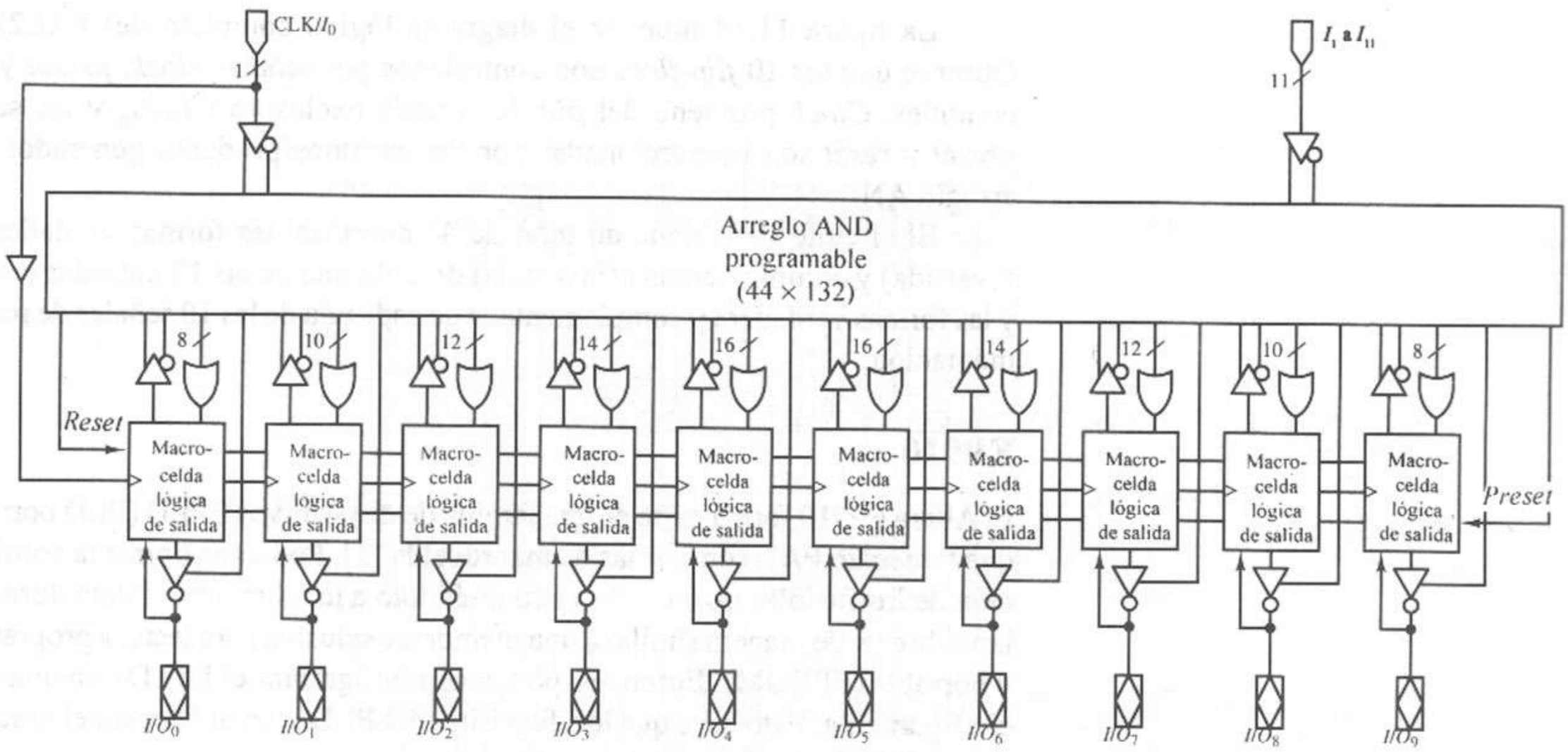


Figura 11.12 PLD PAL22V10 con macroceldas programables [2]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

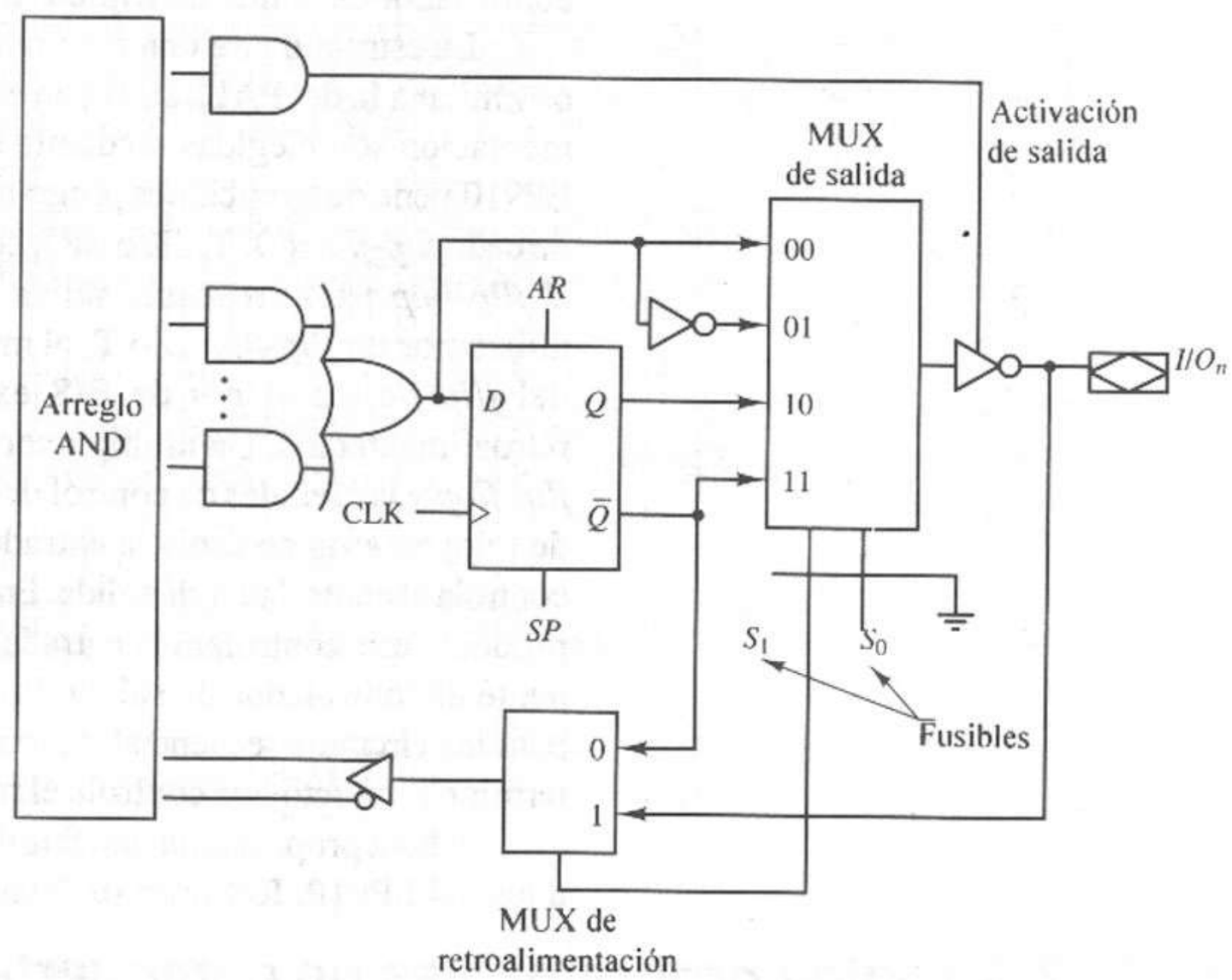


Figura 11.13 Estructura de una macrocelda programable del PAL22V10 [2]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

La figura 11.14 muestra el diagrama lógico completo del PAL22V10. Observe que los 10 *flip-flops* son controlados por señales *clock*, *preset* y *reset* comunes. *Clock* proviene del *pin* de entrada exclusivo CLK/I_0 y las señales *preset* y *reset* son proporcionadas por los términos producto generados en el arreglo AND.

El circuito PAL tiene un total de 44 entradas, las formas verdadera (no invertida) y complementada (invertida) de cada una de las 12 entradas externas y las formas verdadera y complementada de cada una de las 10 señales de retroalimentación.

EP910

El Altera EP910 forma parte de una familia de dispositivos EPLD (PLD borrables) que tienen un PAL con salidas de macrocelda [3]. Podemos borrar la configuración de los fusibles de un EPLD exponiéndolo a una luz ultravioleta durante un lapso breve, de manera similar a una memoria exclusiva para lectura programable y borrable (EPROM). Entonces, podemos reprogramar el EPLD con una nueva configuración. Esto hace que los dispositivos EPLD sean útiles para el desarrollo de prototipos.

La figura 11.15 muestra que el EP910 tiene 24 macroceldas. La sección PAL del EP910 tiene 72 entradas, que provienen de las formas verdaderas y complementadas de 12 *pines* de entrada exclusivos y las líneas de retroalimentación de las 24 macroceldas; además, genera 72 términos producto. Cada una de las 24 macroceldas es controlada por una salida PAL, que es una suma de ocho términos producto. Un término producto adicional controla la entrada *reset* asíncrona del *flip-flop* de la macrocelda, y otro término producto controla la entrada de reloj del *flip-flop* o la línea de control de activación de la salida del controlador de salida de triple estado.

La estructura de una macrocelda de salida, que aparece en la figura 11.16, es similar a la del PAL22V10, en el sentido de que las señales de salida y retroalimentación son elegidas mediante multiplexores programables. Sin embargo, el EP910 tiene otras opciones programables. Podemos programar el modo operativo de cada *flip-flop* (D, T, JK o SR), como muestran las figuras 11.17a a d, o ignorar el *flip-flop* para crear una salida combinatoria, como en la figura 11.17e. Si utilizamos un *flip-flop* D o T, el multiplexor de retroalimentación elige la salida del *flip-flop* o el *pin* de E/S externo para que proporcionen la señal de retroalimentación. Un multiplexor de dos posiciones elige la entrada de reloj del *flip-flop* y las señales de control del controlador de salida. En una posición, el *pin* de reloj externo controla la entrada de reloj del *flip-flop*, y un término producto controla al controlador de salida. En la otra posición, el multiplexor elige el término producto que controlará la entrada de reloj del *flip-flop* y activa en forma permanente el controlador de salida. Por lo general, el *pin* de reloj externo se utiliza para los circuitos secuenciales síncronos, mientras que la capacidad de utilizar un término producto que controle el reloj es necesaria para una operación asíncrona.

Altera proporciona una familia de EPLD cuyas características son similares a las del EP910. Resumimos éstas en la tabla 11.5.

11.2 Arreglos programables de compuertas

Por su arquitectura básica, los PLD están limitados a la realización de expresiones de conmutación SOP de dos niveles, para todas las variables de salida y de excitación. Además, la cantidad y las configuraciones de los registros también

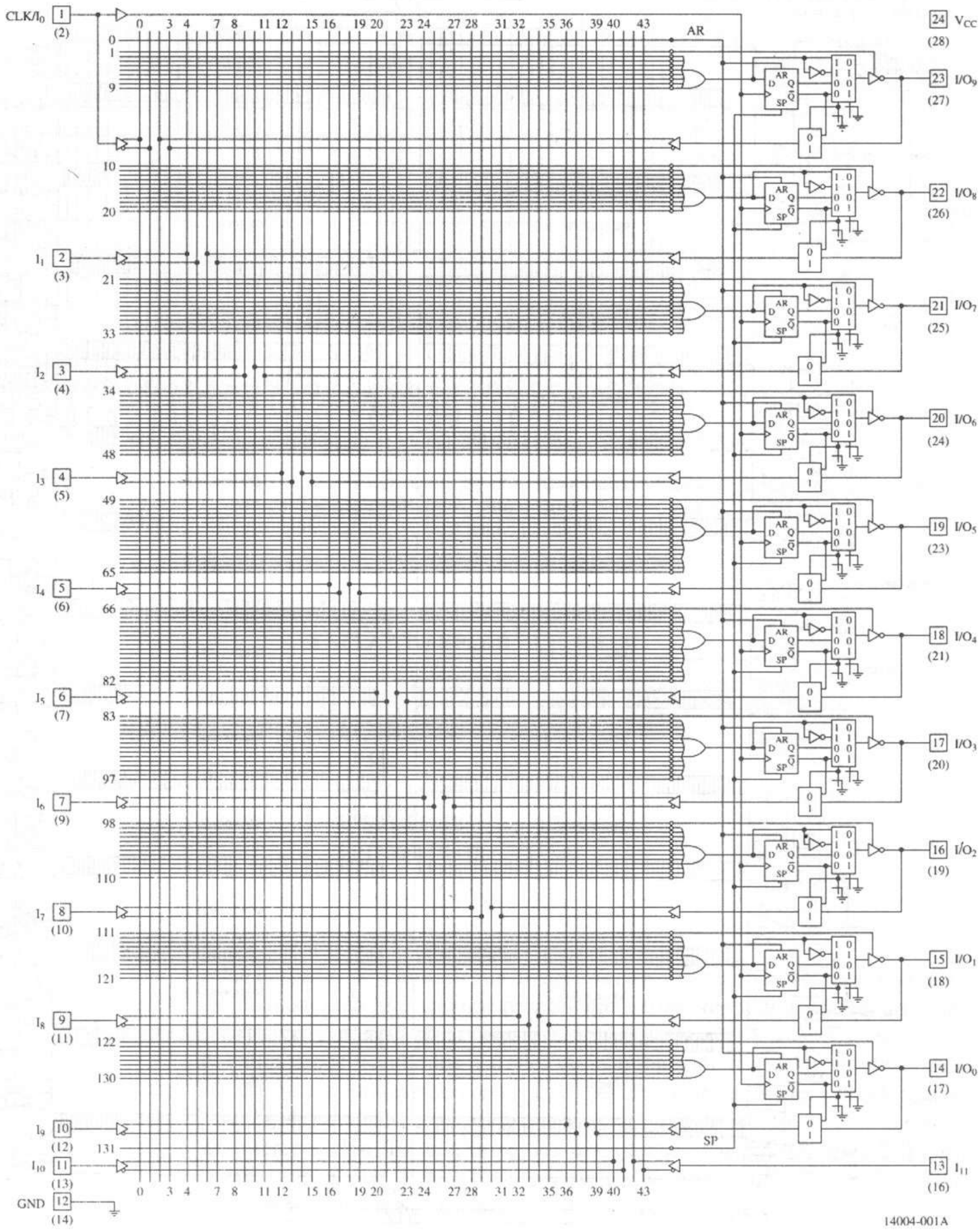


Figura 11.14 Diagrama lógico completo del PLD PAL22V10 [2]. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

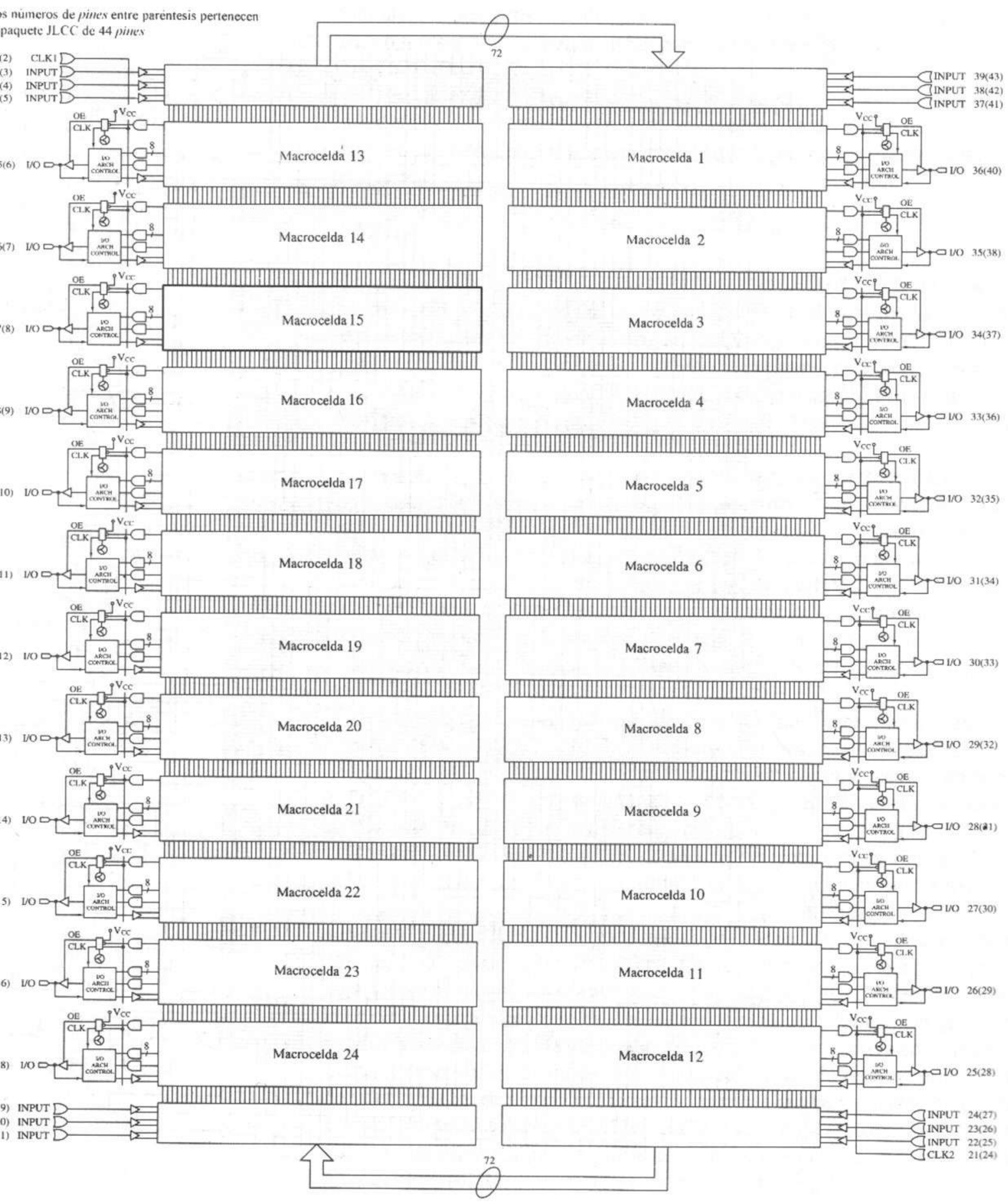


Figura 11.15 EPLD Altera EP910 [3]. *Fuente*: Altera, User-compatible Logic Databook, Altera Corp., Santa Clara, CA, 1988.

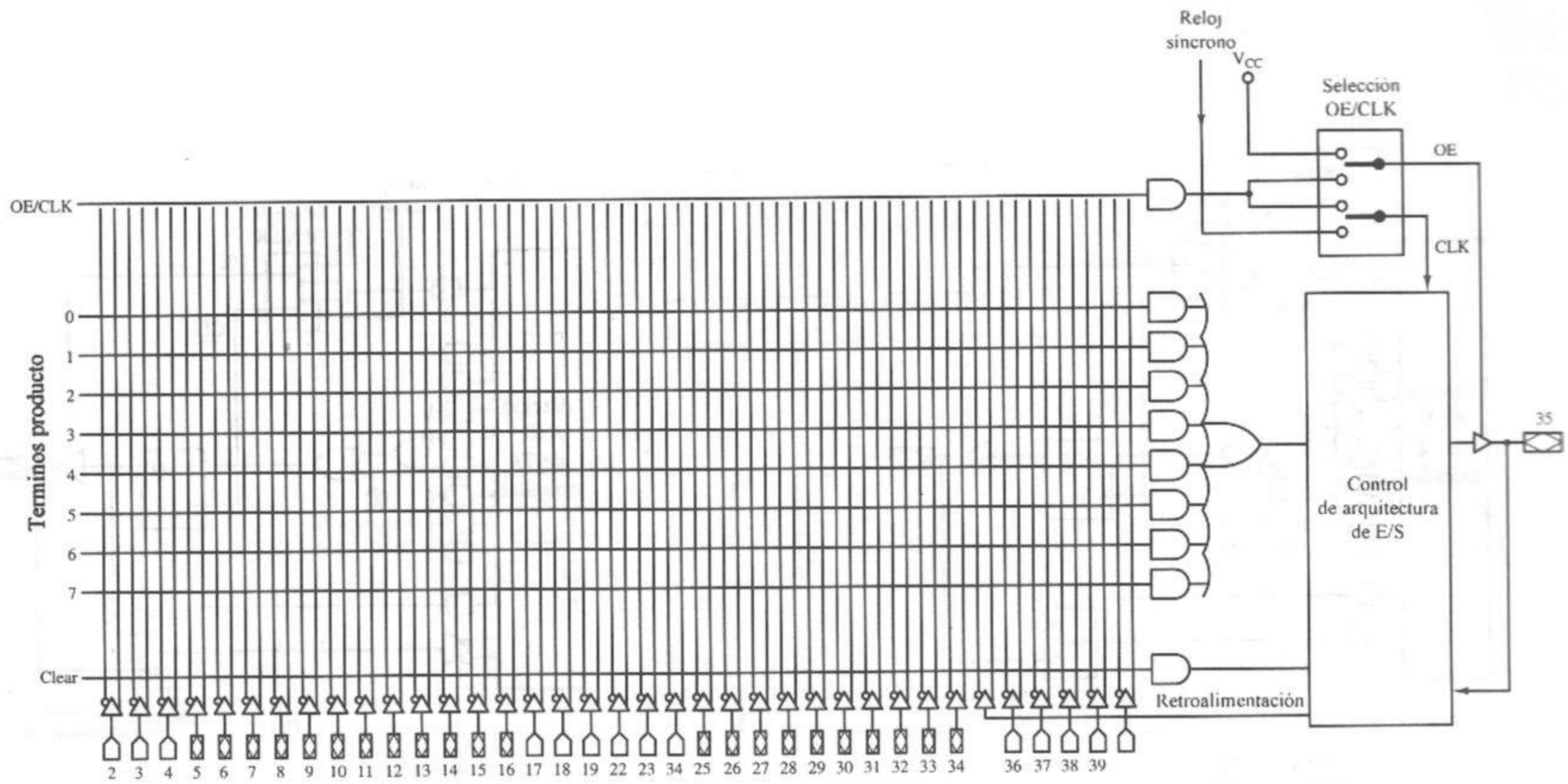


Figura 11.16 Estructura de una macrocelda del EP910 [3]. *Fuente*: Altera, User-compatible Logic Databook, Altera Corp., Santa Clara, CA, 1988.

TABLA 11.5 EPLDS ALTERA [3]

Dispositivo	Macroceldas	Entradas
EP310, EP320	8	10
EP512	12	10
EP600, EP610	16	4
EP900, EP910	24	12
EP1800, EP1810	48	16

están limitadas. En contraste, un *arreglo de compuertas* no tiene una arquitectura de interconexión fija, sino que proporciona varias compuertas lógicas sencillas y otros elementos sin conexiones fijas entre sus entradas y salidas. Creamos los circuitos especificando los patrones de interconexión deseados, prácticamente sin establecer límites sobre estos patrones. Podemos solicitar a un fabricante arreglos de compuertas y pedirle que los configure, proporcionándole el patrón de interconexión. En contraste, los *arreglos de compuertas programables mediante campos* (FPGA) son programados por el usuario mediante un programador de dispositivos especial que crea las interconexiones entre las compuertas.

11.2.1 Arreglos de celdas lógicas

El *arreglo de celdas lógicas* (LCA) es una familia única de dispositivos programables presentados por Xilinx [4]. La figura 11.18 muestra que cada LCA

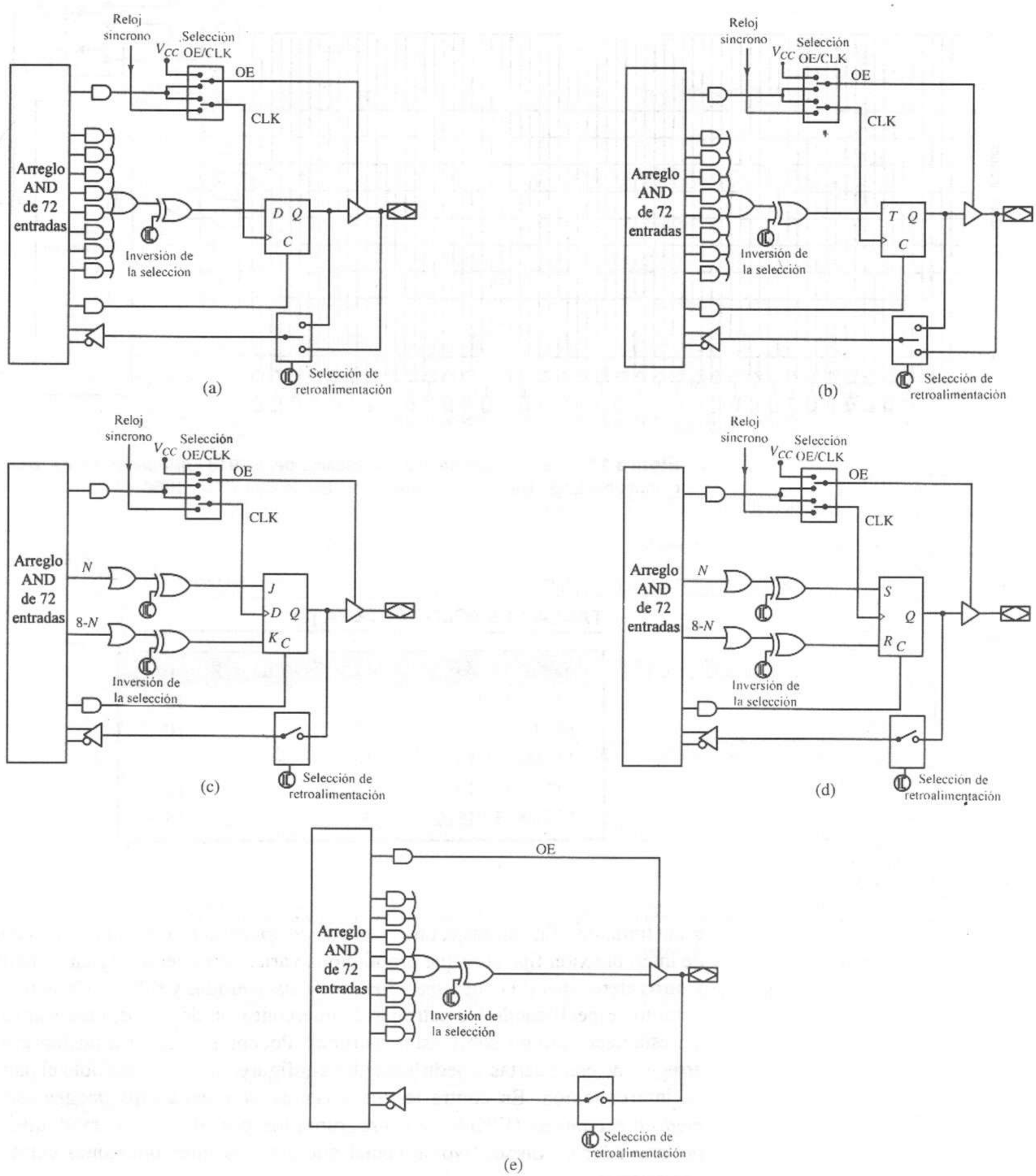


Figura 11.17 Arquitecturas de salida del EP910 [3]. **(a)** *Flip-flop* D. **(b)** *Flip-flop* T. **(c)** *Flip-flop* JK. **(d)** *Flip-flop* SR. **(e)** Omisión del *flip-flop*. *Fuente*: Altera, User-compatible Logic Databook, Altera Corp., Santa Clara, CA, 1988.

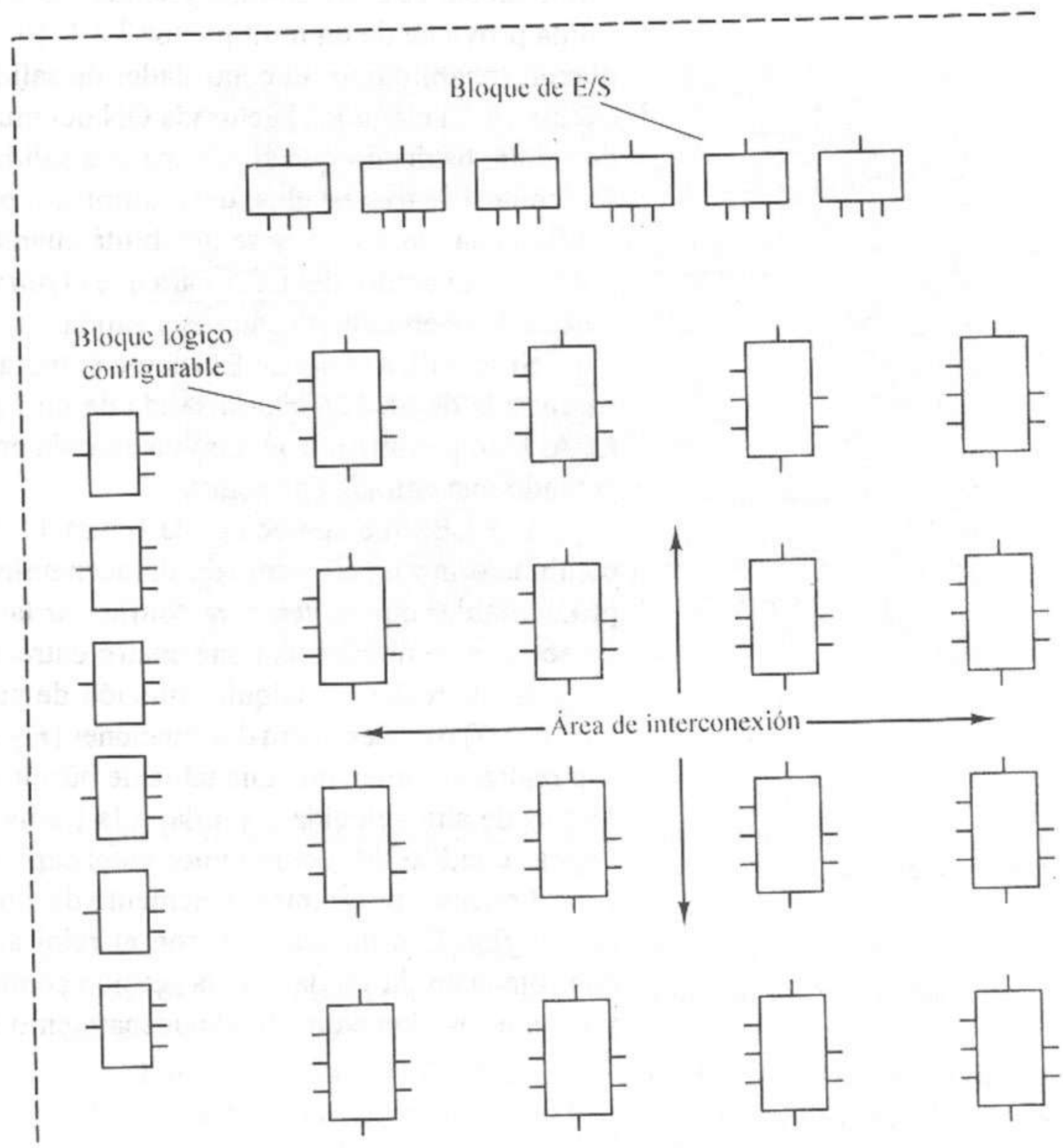


Figura 11.18 Estructura del arreglo de celdas lógicas de Xilinx [4]. Figura cortesía de Xilinx, Inc. © Xilinx, Inc., 1992. Todos los derechos reservados.

tiene una matriz de *bloques lógicos configurables* (CLB), cada uno de los cuales tiene un *flip-flop* y un pequeño bloque de lógica combinatoria. La matriz de CLB está rodeada por *bloques de E/S configurables por el usuario* (IOB), los cuales ofrecen varias opciones en cada *pin* de E/S. Las entradas y salidas de los CLB e IOB se interconectan mediante segmentos de cable que están en los canales de cableado entre las filas y columnas de bloques. Los multiplexores en las entradas y salidas de los bloques conectan los bloques con estos segmentos de cable, y las matrices de conmutadores en las intersecciones de las filas y columnas conectan los segmentos de cable.

Creamos un circuito en un LCA especificando la configuración de cada CLB e IOB, así como las interconexiones entre estos bloques. Programamos la configuración en el LCA cargando un patrón de unos y ceros en una memoria volátil de acceso aleatorio (RAM) dentro del LCA cada vez que encendemos el circuito. Si así se desea, se pueden hacer cambios a la configuración mientras el LCA está operando, lo que produce un diseño con cambios dinámicos.

La figura 11.19 muestra la estructura del IOB. Podemos programar cada *pin* de E/S para que sea una entrada o salida exclusiva o para que alterne en forma dinámica entre entrada y salida. La señal de control del controlador de la salida proviene de un multiplexor 3 a 1. Si elegimos la entrada OFF del multiplexor, inhabilitamos el controlador de salida y el *pin* opera como una entrada exclusiva. Si elegimos la entrada ON del multiplexor, se habilita el controlador de salida, haciendo que el *pin* sea una salida exclusiva. Si elegimos la entrada *TS* (control de tres estados) del multiplexor para controlar el controlador, éste se habilita cuando $TS = 0$ y se inhabilita cuando $TS = 1$. La señal *TS* es generada por la lógica dentro del LCA para que el *pin* alterne en forma dinámica entre los modos de operación de entrada y salida.

Si se utiliza el *pin* de E/S como entrada, un multiplexor 2 a 1 elige la señal de entrada de un *buffer* o la salida de un *flip-flop* para suministrar la señal al LCA. Esto permite introducir una entrada por reloj a un *flip-flop* y conservarla, creando una entrada con registro.

El CLB, que aparece en la figura 11.20, contiene una sección de lógica combinatoria y un elemento de almacenamiento, junto con varios multiplexores programables que sirven para configurar las opciones de salida y del *flip-flop*. La sección combinatoria tiene cuatro entradas (A, B, C y D) y dos salidas (F y G) y puede realizar cualquier función de cuatro variables con una sola salida (con $F = G$) o cualesquiera dos funciones (F y G) de tres variables. Estas funciones son realizadas mediante una tabla de búsqueda almacenada en una memoria de 16 bits de alta velocidad, similar a la forma en que se implantan las funciones lógicas con PROM, como vimos en el capítulo 5.

Podemos programar el elemento de almacenamiento para que opere como un *flip-flop* D o un *latch* D, con el reloj activo alto o bajo. La entrada D es controlada por la salida F de la sección combinatoria. Las entradas del reloj, *set* y *reset* a los elementos de almacenamiento son elegidas por multiplexores. El

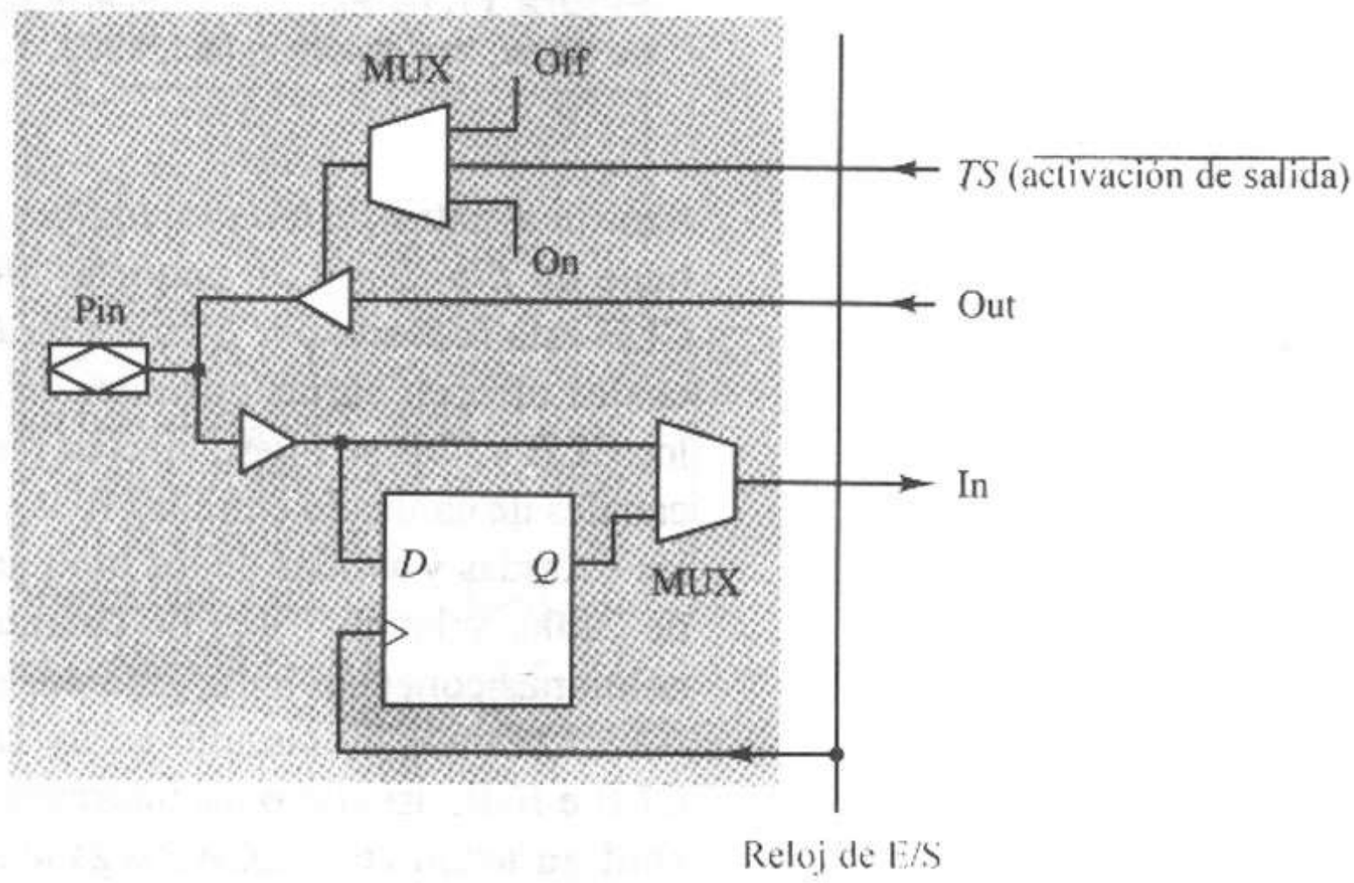


Figura 11.19 Estructura del bloque de E/S del LCA [4]. Figura cortesía de Xilinx, Inc. © Xilinx, Inc., 1992. Todos los derechos reservados.

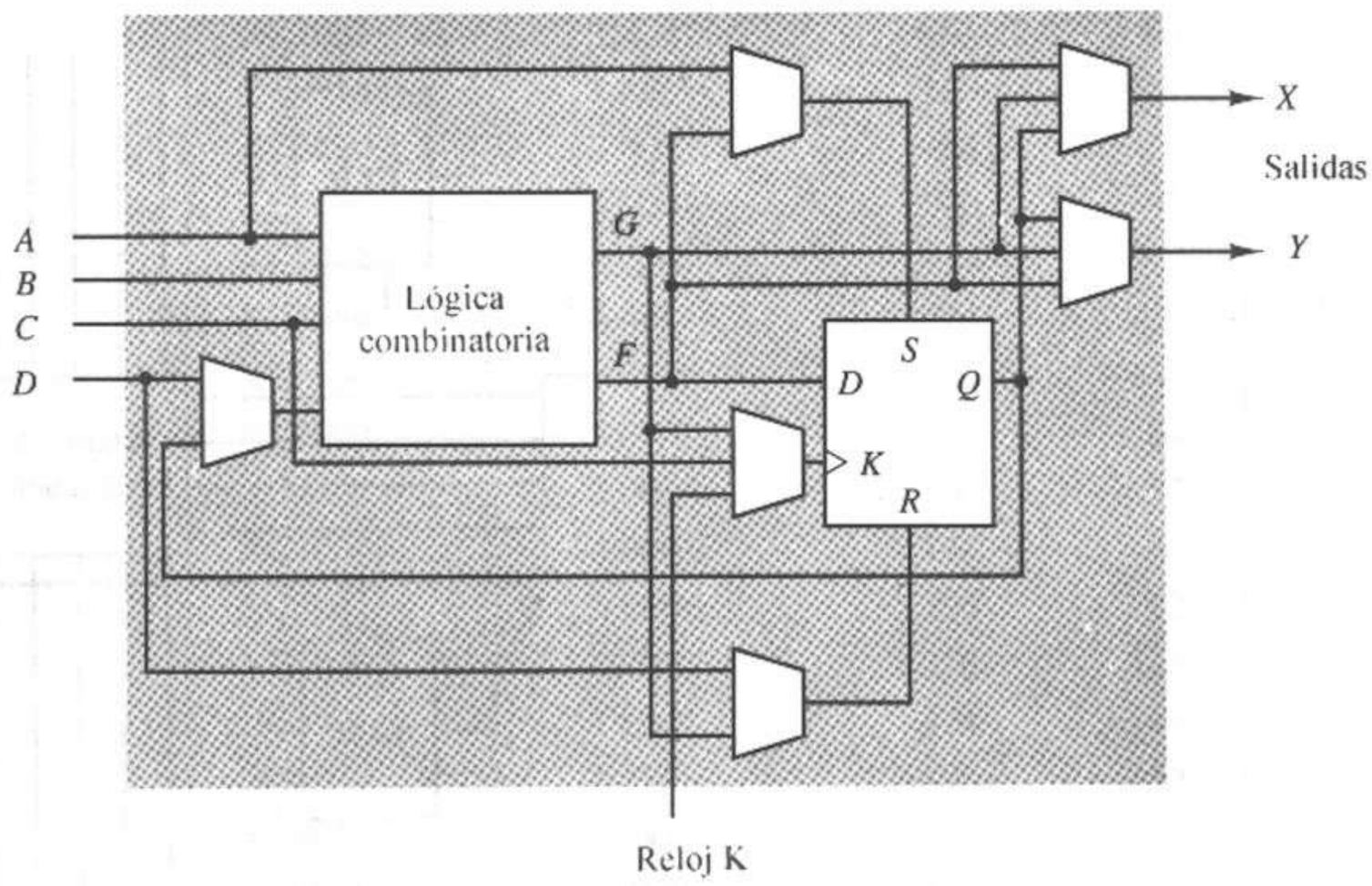


Figura 11.20 Estructura del bloque lógico configurable (CLB) del LCA [4]. Figura cortesía de Xilinx, Inc. © Xilinx, Inc., 1992. Todos los derechos reservados.

reloj puede ser síncrono, con la señal de reloj distribuida por el *pin K* común a todos los CLB; o asíncrono, con el reloj suministrado por el bloque de función combinatoria o la entrada *C*.

Las dos salidas del CLB, *X* y *Y*, son seleccionadas por multiplexores de salidas separadas cuyas entradas son las salidas *F* y *G* de la sección combinatoria y la salida del *flip-flop Q*. Así, el CLB puede tener salidas combinatorias, con registro o ambas.

Las entradas y salidas de los CLB y de los IOB están conectadas a segmentos de metal (cables) que se encuentran en los canales entre las filas y columnas de los bloques. Cada intersección fila/columna es una matriz de conmutación programada para conectar los elementos lógicos deseados. Las interconexiones se hacen mediante tres tipos de recursos de ruteo: interconexiones de propósito general, líneas largas y conexiones directas.

Las *interconexiones de propósito general* se encuentran en los canales entre las filas y las columnas de los CLB y los IOB, como muestra la figura 11.21. En cada intersección fila/columna hay una matriz de conmutación que conecta segmentos de cables de las filas y columnas conectadas. Hay cuatro segmentos de cable horizontales entre las filas y cinco entre las columnas; cada segmento recorre la longitud del canal entre las matrices de conmutación.

Las *líneas largas* se extienden a través de todo el arreglo CLB, con dos líneas por canal vertical y uno por fila, como muestran las líneas gruesas en la figura 11.21. Éstas sirven para distribuir los relojes y otras señales con una desviación mínima dentro del circuito.

Hay *interconexiones directas* separadas entre CLB adyacentes. La salida *X* de cada CLB puede conectarse en forma directa a las entradas *C* y *D* del CLB inmediato superior o a las entradas *A* y *B* del CLB inmediato inferior. La salida

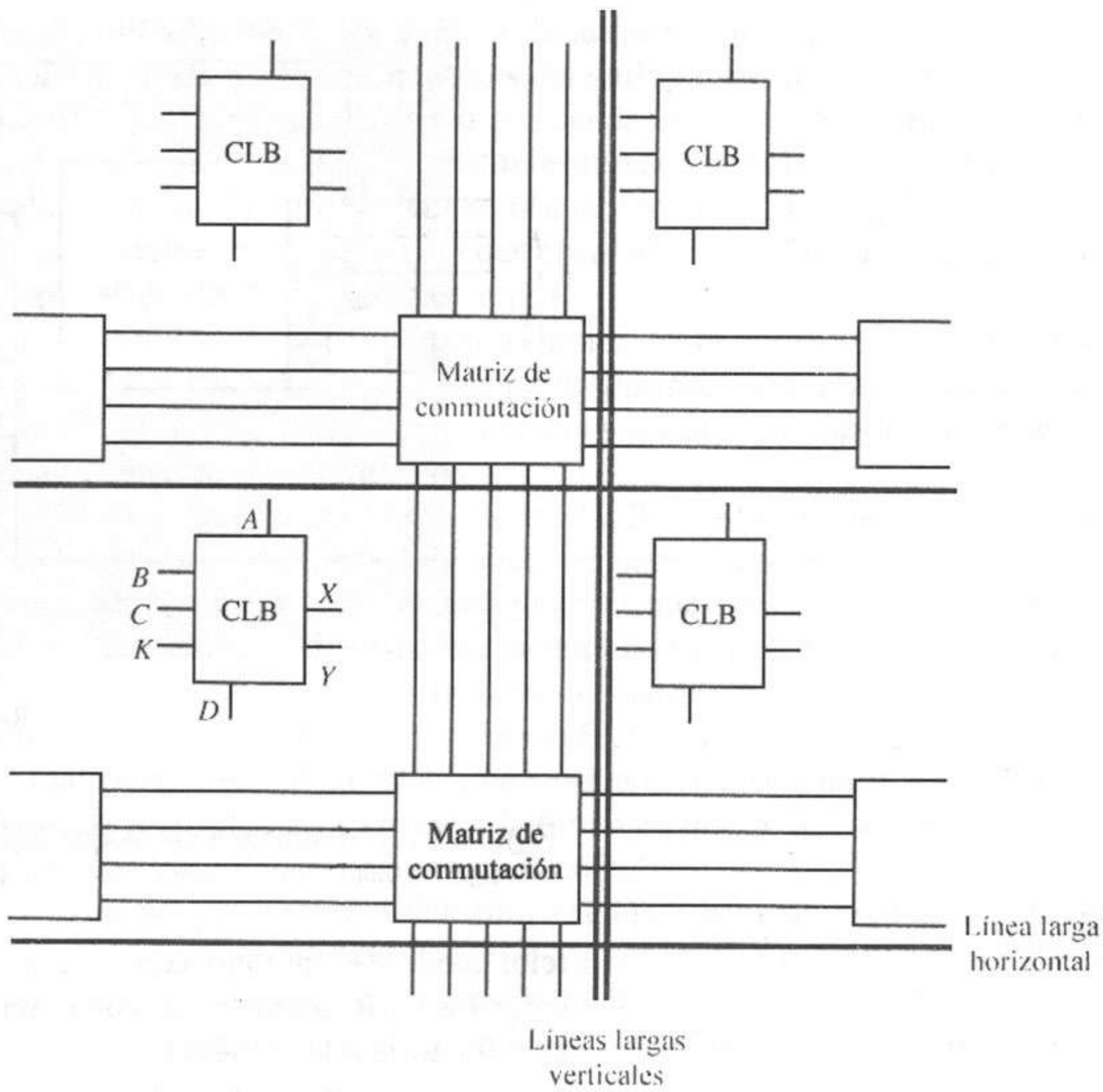


Figura 11.21 Interconexiones de línea larga y de propósito general en LCA [4]. Figura cortesía de Xilinx, Inc. © Xilinx, Inc., 1992. Todos los derechos reservados.

Y de un CLB puede conectarse en forma directa a la entrada *B* del CLB inmediato a su derecha. Así, al organizar el diseño de un LCA, el flujo de la señal es por lo general de izquierda a derecha en forma horizontal y del centro hacia fuera en forma vertical.

Los dispositivos de una familia de LCA Xilinx comparten una arquitectura CLB o IOB común, pero tienen distintas cantidades de CLB e IOB, como se resume en la tabla 11.6. Existen varias familias; las series XC3000 y XC4000 tienen cierta funcionalidad adicional agregada a los CLB y IOB. Las configuraciones CLB e IOB presentadas hasta ahora pertenecen a la serie XC2000, que incluye el XC2064, con 58 IOB y 64 CLB dispuestos en una matriz 8 × 8, y el XC2018, con 74 IOB y 100 CLB ordenados en una matriz 10 × 10. El XC2064 equivale a aproximadamente 1200 compuertas lógicas, mientras que el XC2018 equivale a 1800 compuertas, aproximadamente.

Las capacidades de los dispositivos de la serie XC3000 varían entre 2000 y 9000 compuertas lógicas equivalentes. El CLB de la XC3000 extiende el diseño CLB de los dispositivos de la serie XC2000 a un bloque de función combinatoria de cinco entradas y dos *flip-flops*, junto con una sección de control para elegir varias opciones de *flip-flop* y de salida.

TABLA 11.6 DISPOSITIVOS DE ARREGLOS DE CELDAS LÓGICAS DE XILINX [4]

Dispositivo	Compuertas equivalentes	CLB	IOB
XC2064	1,200	64	64
XC2018	1,800	100	64
XC3020	2,000	64	64
XC3030	3,000	100	80
XC3042	4,200	144	96
XC3064	6,400	224	120
XC3090	9,000	320	144
XC4002	2,000	64	64
XC4003	3,000	100	80
XC4004	4,000	144	96
XC4005	5,000	196	112
XC4006	6,000	256	128
XC4008	8,000	324	144
XC4010	10,000	400	160
XC4013	13,000	576	192
XC4016	16,000	676	208
XC4020	20,000	900	240

La familia más reciente de dispositivos, la serie XC4000, puede tener matrices desde 8 × 8 hasta 30 × 30 de CLB y desde 64 hasta 240 IOB, proporcionando el equivalente de 2000 a 20,000 compuertas lógicas. La arquitectura del CLB de la familia XC4000 representa un gran avance con respecto de las familias XC2000 y XC3000. Como muestra la figura 11.22, este CLB tiene dos elementos de almacenamiento independientes y tres generadores de función, con un total de 13 entradas y 4 salidas. Dos salidas son combinatorias y dos tienen registros. Dos generadores de función tienen cuatro entradas (F_1 a F_4 y G_1 a G_4), mientras que las tres entradas del tercer generador de función incluyen las salidas de los otros dos generadores de función (F' y G') y una entrada externa H_1. Como en el caso de las familias XC2000 y XC3000, se utiliza memoria de búsqueda de alta velocidad para implantar los generadores de función. Juntos, los tres generadores de función pueden generar cualesquiera dos funciones independientes de cuatro variables, cualquier función de cinco variables una función de cuatro variables, y algunas funciones de cinco variables, o algunas funciones de hasta nueve variables. Así, con un único CLB podemos realizar funciones combinatorias de más entradas que con las familias LCA anteriores.

La figura 11.22 no muestra un circuito de acarreo aritmético especial entre los dos generadores de función F y G. Este circuito permite crear un sumador de 2 bits de manera conveniente, con el acarreo entre los módulos controlado por la lógica de acarreo especial.

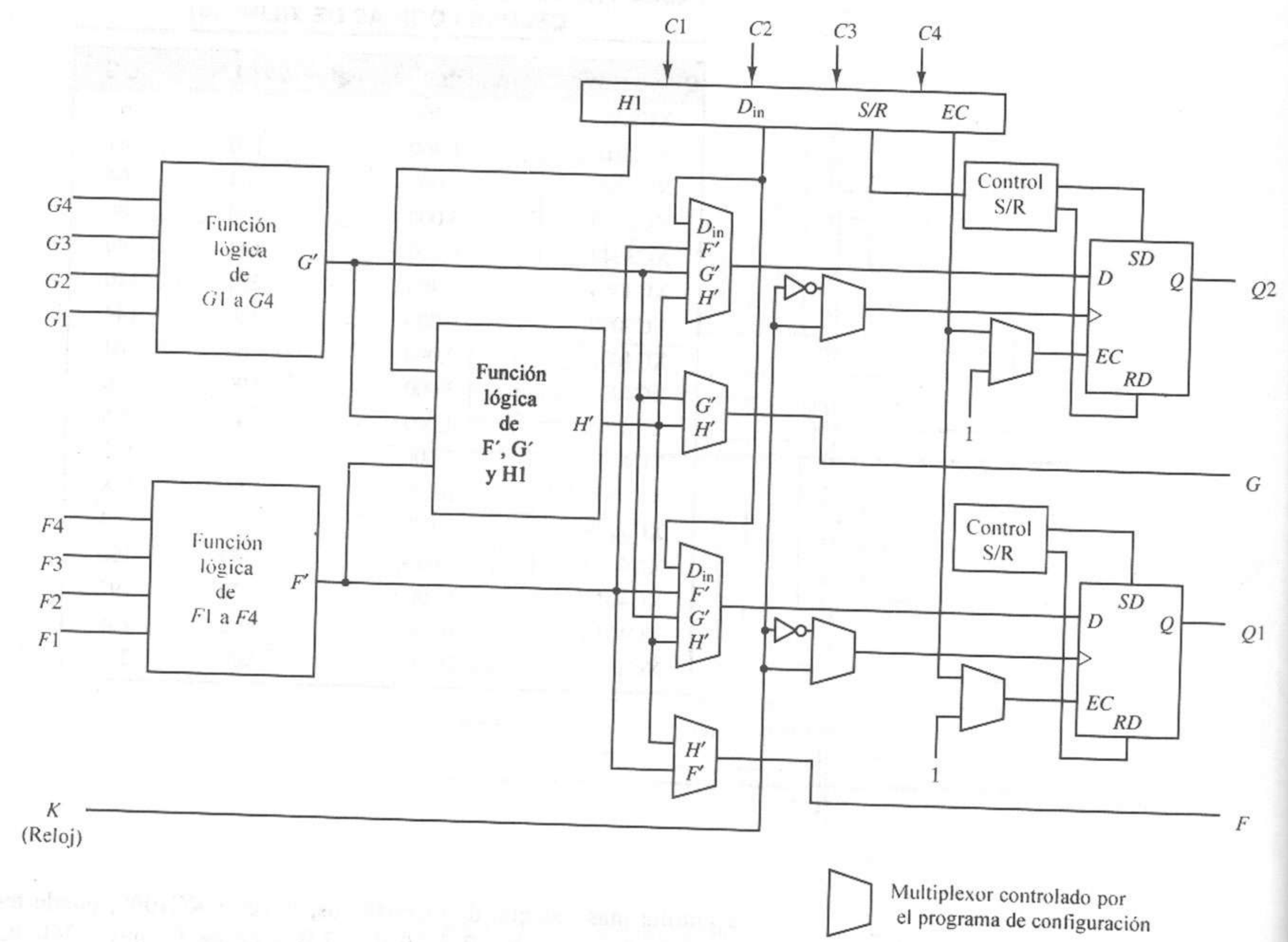


Figura 11.22 Arquitectura del CLB XC4000 de Xilinx [4]. Figura cortesía de Xilinx, Inc. © Xilinx, Inc., 1992. Todos los derechos reservados.

Los dos elementos de almacenamiento del CLB de la familia XC4000 son *flip-flops* D disparados por transición, con una entrada de reloj externa común *K*. Cada *flip-flop* se programa de manera independiente para dispararse en la transición ascendente o descendente del reloj. A fin de facilitar la retención del estado de un circuito, cada *flip-flop* utiliza entradas de activación de reloj individuales (*EC*). Además, cada *flip-flop* tiene una entrada independiente que puede programarse como una línea de control *set* o *reset*, de modo que los dos *flip-flops* queden configurados de manera independiente. Observe que las entradas *DIN*, *S/R* y *EC* se conectan con las entradas CLB externas C_1 a C_4 mediante un multiplexor. Esto permite conectar cada una de las cuatro entradas en cualquier orden.

Además de los diseños CLB y IOB mejorados, la XC4000 también incluye el doble de interconexiones de líneas de las familias anteriores, además de un número mayor de otros recursos de ruteo.

11.2.2 FPGA ACT

La familia ACT-1 de FPGA de Actel [5] también está construida en torno de una matriz de módulos lógicos rodeados por bloques de E/S. Los canales de cableado entre las filas contienen 25 cables cada uno y se tienen 13 pistas de cableado verticales por cada columna de módulos lógicos.

La figura 11.23 muestra que cada módulo lógico tiene ocho entradas y una salida y puede realizar las cuatro compuertas lógicas básicas (AND, OR, NAND y NOR) con dos, tres o cuatro entradas cada una y entradas activas altas o activas bajas. También se pueden realizar *latches* D, compuertas XOR y funciones simples AND-OR y OR-AND con un único módulo lógico. Todas las funciones de conmutación son realizadas con multiplexores 2 a 1, según lo descrito en el capítulo 4, de modo que cuatro de las entradas del módulo lógico se aplican a las entradas de datos del multiplexor y las otras cuatro controlan las líneas de selección del multiplexor, como muestra la figura 11.23. Cada entrada de módulo se conecta a uno de los canales de cableado vertical, lo mismo que la salida del módulo. Como en los LCA de Xilinx, un bloque lógico en cada *pin* de E/S permite configurarlo como entrada exclusiva, salida exclusiva o señal bidireccional.

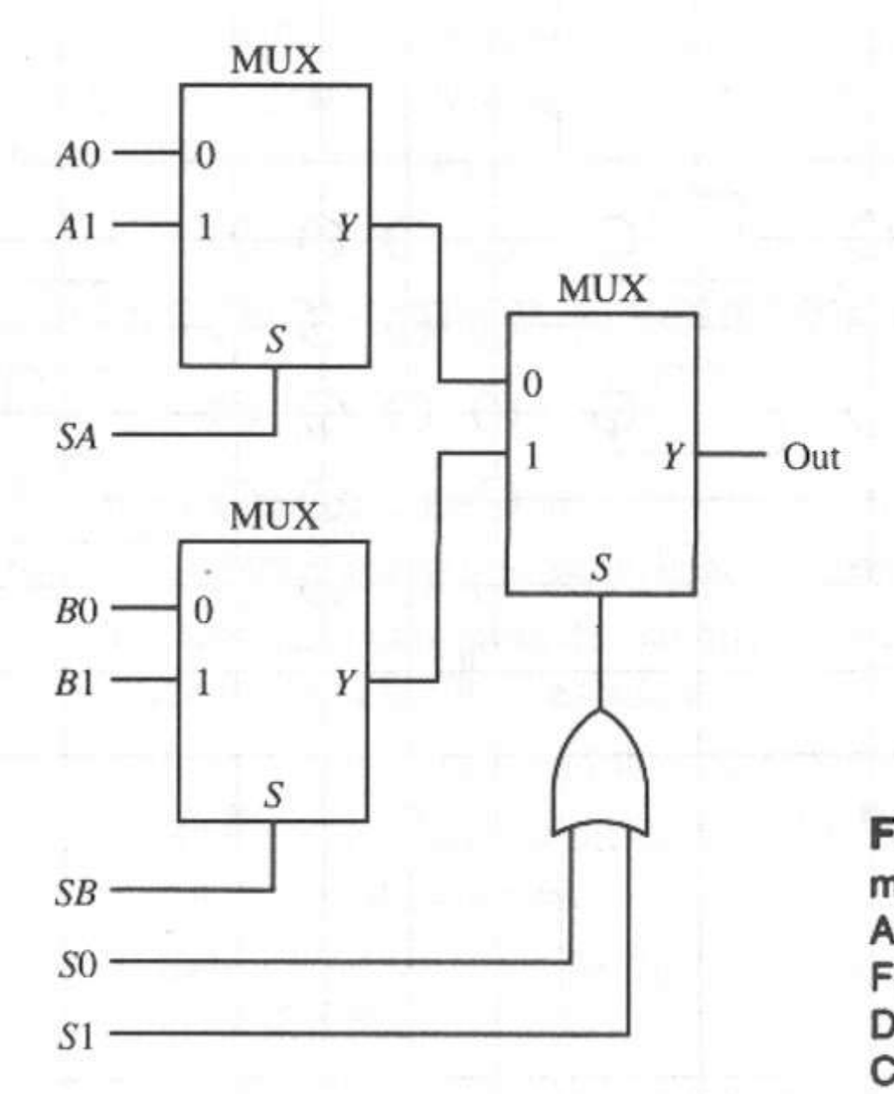


Figura 11.23 Estructura de un módulo lógico de la familia ACT-1 de Actel [5]. *Fuente:* Actel, "ACT Family Field Programmable Gate Array Databook", Actel Corp., Sunnyvale, CA, 1991.

Los módulos lógicos de los FPGA Act no tienen circuitos de *flip-flop* exclusivos. Se puede realizar un *latch* D con un único bloque de lógica combinatoria, usando la retroalimentación, y realizar un *flip-flop* con dos bloques lógicos. Así, otros *flip-flops* y compuertas más complejas se implantan conectando varios módulos lógicos entre sí. El sistema de diseño Actel incluye una biblioteca de circuitos comunes, llamados *macros*, que se pueden implantar mediante los módulos lógicos. Los *macros duros* son circuitos que se pueden realizar con un único módulo lógico, mientras que los *macros suaves* requieren más de un módulo lógico para su realización. Cada macro suave comprende

configuraciones para dos o más bloques lógicos y un patrón de interconexión entre los bloques.

La figura 11.24 muestra la arquitectura de interconexión. Para simplificar la exposición, sólo mostramos algunas de las líneas de interconexión. Las pistas de cableado vertical recorren los propios bloques lógicos, mientras que las pistas de cableado horizontal están entre las filas de módulos lógicos. Cada entrada y salida de un módulo lógico se conecta a un segmento de cable vertical exclusivo. Algunos segmentos verticales sirven como alimentadores entre los canales. Las conexiones se hacen con los segmentos de cable horizontales mediante fusibles programables que se colocan en cada intersección de un segmento vertical y horizontal (aparecen como círculos en la figura 11.24). Los segmentos horizontales se conectan eliminando los fusibles de la intersección. Así, se crea la estruc-

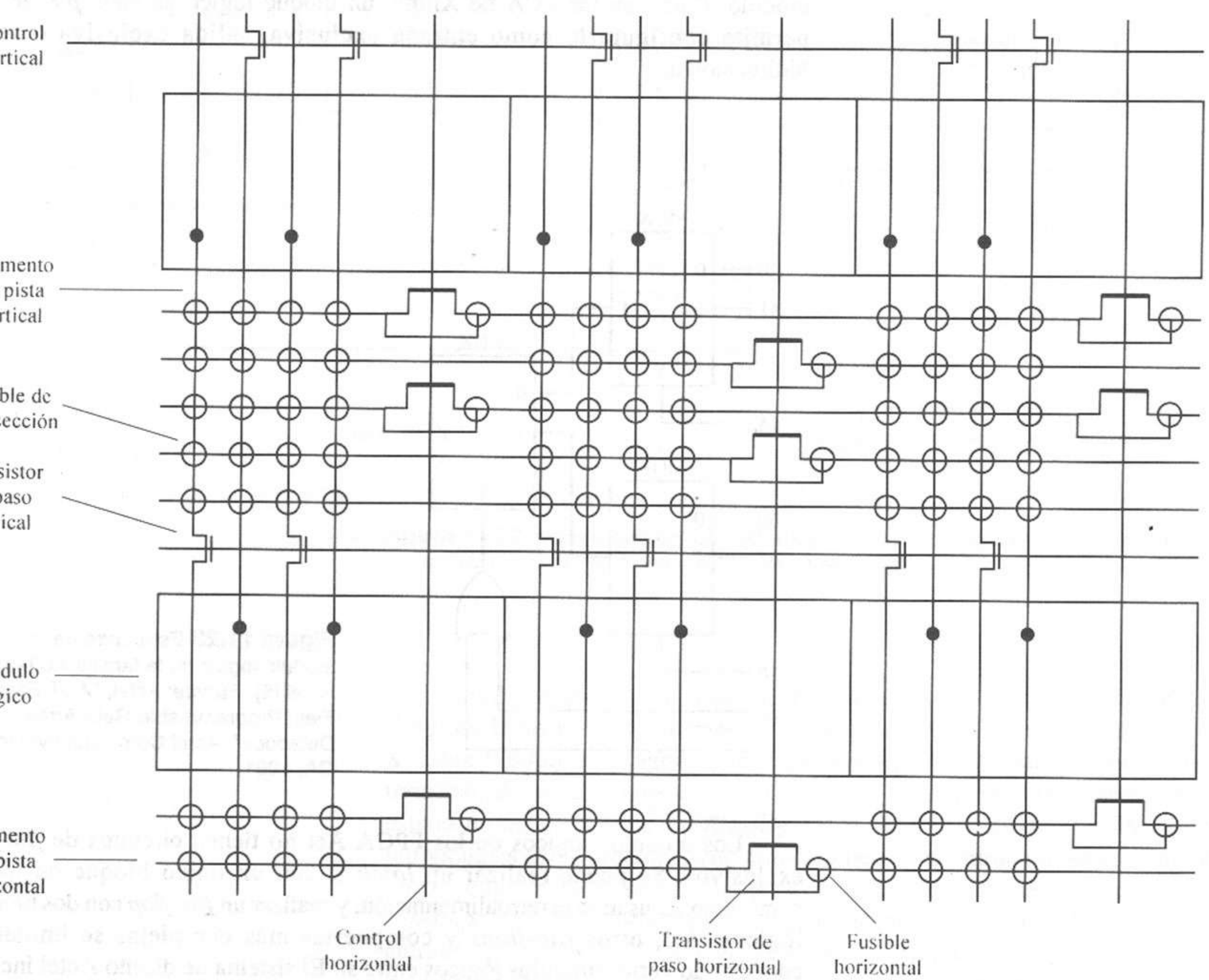


Figura 11.24 Arquitectura de interconexión del ACT-1 de Actel [5]. *Fuente:* Actel, "ACT Family Field Programmable Gate Array Databook", Actel Corp., Sunnyvale, CA, 1991.

tura de circuito deseada especificando un patrón de fusibles que es preciso eliminar para crear las interconexiones necesarias entre los módulos lógicos.

Los dispositivos de la familia ACT-1 difieren en el número de módulos lógicos disponibles y los *pines* de E/S, como muestra la tabla 11.7. Por ejemplo, el A1010 contiene 295 módulos lógicos y 57 *pines* de E/S, y equivale a aproximadamente 1200 compuertas lógicas, mientras que el A1020 contiene 546 módulos lógicos y 69 *pines* de E/S, y equivale a aproximadamente 2000 compuertas.

TABLA 11.7 DISPOSITIVOS DE LAS FAMILIAS ACT-1 (A10XX) Y ACT-2 (A12XX) DE ACTEL [5]

Dispositivo	Compuertas equivalentes	Módulos lógicos	E/S
A1010	1200	295	57
A1020	2000	546	69
A1225	2500	451	82
A1240	4000	684	104
A1280	8000	1232	140

La familia ACT-2 incluye dispositivos con muchas más compuertas lógicas y más *pines* de E/S, que varían desde 2500 compuertas lógicas para el A1225 hasta 8000 compuertas para el A1280 (1232 módulos lógicos y 104 *pines* de E/S).

11.3 Diseño de circuitos secuenciales y selección de dispositivos PLD

El diseño de un circuito secuencial con una lógica programable se efectúa de la misma manera que cuando se utilizan compuertas discretas y *flip-flops*.

1. Diseñe un diagrama de estados a partir de la descripción del problema y deduzca la tabla de estados.
2. Identifique y elimine los estados redundantes.
3. Realice una asignación de variables de estado y deduzca una tabla de transición de estados.
4. Seleccione los tipos de *flip-flop* y deduzca las tablas de excitación para cada *flip-flop*.
5. Deduzca las ecuaciones de excitación a partir de las tablas de excitación.
6. Deduzca las ecuaciones de salida a partir de la tabla de estados.
7. Asocie las ecuaciones con compuertas lógicas y *flip-flops*.

Los pasos 1 a 3 son independientes del hecho de que se utilicen compuertas lógicas o dispositivos lógicos programables para la implantación; estos pasos se realizan como en el capítulo 8. En el paso 4, la elección del tipo de *flip-flop* puede estar dictada por el tipo de dispositivo lógico por utilizar, o viceversa. En algunos dispositivos FPLS y PLD con registros, el tipo de *flip-flop* está fijo. En estos casos, debemos trabajar con los tipos dados. En los dispositivos FPLS, como el PLS155 y en algunos dispositivos PAL con macroceldas de salida

programables, cada tipo de *flip-flop* se puede programar en forma individual, en cuyo caso podemos proceder como si dispusiéramos de *flip-flops* discretos arbitrarios y realizar la elección que mejor se ajuste al problema.

La deducción de las ecuaciones de excitación y de salida también puede estar restringida por el tipo de dispositivo. En los dispositivos FPLS y PLD debemos utilizar formatos de suma de productos de dos niveles para asociar las ecuaciones con un arreglo AND/OR programable. En los dispositivos LCA el método original consiste en ajustar la lógica de excitación de cada *flip-flop* al mismo CLB que contiene el *flip-flop*. En los dispositivos FPGA, existen menos restricciones sobre la arquitectura del *flip-flop* y su lógica de excitación, de modo que el diseñador se puede concentrar en la minimización del número de celdas utilizadas.

Una vez desarrolladas todas las ecuaciones, la realización del diseño en el dispositivo elegido requiere la asociación de las ecuaciones lógicas con los recursos del dispositivo elegido. Esto se realiza por lo general con el apoyo de herramientas de CAD especiales que convierten las ecuaciones al formato adecuado y las ajustan al dispositivo objetivo, creando un mapa de fusibles que puede cargarse en el dispositivo mediante un programador de dispositivos.

La elección de un dispositivo lógico programable para realizar un diseño dado está determinada por varias características fundamentales del diseño.

1. *Número de entradas*. El diseño tendrá una o más entradas "dato" externas, así como un reloj y tal vez señales de control *set/reset*. Las entradas de un dispositivo programable incluyen líneas de entrada exclusivas que controlan el arreglo AND/OR, líneas de E/S que pueden programarse para operar como entradas y otras entradas exclusivas que controlan las entradas de reloj del *flip-flop*, las entradas *set/reset*, etcétera.

2. *Número de elementos de almacenamiento*. El número de elementos de almacenamiento necesarios para un diseño depende del número de variables de estado y del número de salidas síncronas necesarias. Las variables de estado se deben realizar en los *flip-flops* cuyas salidas se retroalimentan al arreglo AND/OR. Estos *flip-flops* pueden ser salidas con registro o *flip-flops* ocultos. Las salidas síncronas se deben realizar en salidas con registro, aunque no necesariamente se retroalimenten al arreglo AND/OR. Si se necesita las salidas con registro, se pueden utilizar salidas exclusivas o líneas de E/S configuradas para operar como salidas.

3. *Tipos de flip-flops*. Los tipos de *flip-flops* disponibles en un dispositivo programable pueden ser fijos, en cuyo caso el diseñador debe decidir si los tipos disponibles son adecuados para el diseño. En caso contrario, hay que elegir otro dispositivo. En muchos dispositivos programables se puede programar el modo operativo. En estos casos, el diseñador tiene un máximo de flexibilidad.

4. *Número de salidas*. Un diseño puede necesitar salidas combinatorias o con registro. En la mayor parte de los casos, las salidas están separadas de las variables de estado, aunque en algunos diseños de máquinas Moore las variables de estado también son salidas externas del circuito. Los dispositivos programables pueden tener una o más salidas exclusivas, ya sea con registro o combinatorias, o líneas de E/S que se pueden programar para operar como salidas. Otras opciones de salida pueden incluir una polaridad programable.

5. *Lógica combinatoria.* En los dispositivos basados en PAL y PLA hay que deducir todas las ecuaciones de excitación y de salida en forma de sumas de productos de dos niveles que se puedan asociar con arreglos AND/OR. En los PAL con registros el arreglo OR es fijo, lo que limita el número de términos producto que se pueden utilizar en cada ecuación de excitación y de salida. En los dispositivos FPLS el arreglo OR es programable, lo que permite utilizar un número arbitrario de términos producto en cada ecuación, dentro de los límites del número total de líneas de productos del dispositivo. También se pueden compartir los términos producto entre las expresiones para ahorrar líneas de productos.

6. *Características especiales.* Muchos dispositivos programables tienen varias características especiales que podrían ser recomendables o necesarias para un diseño dado. Entre ellas se encuentran la temporización asíncrona de los *flip-flops*, los términos lógicos para el control de las entradas *set* y *reset* del *flip-flop*, y la lógica de activación para los controladores de salida de triple estado.

7. *Propiedades físicas.* En muchos casos, los diseños están restringidos por ciertas condiciones físicas, como el tamaño del paquete del dispositivo, el consumo de energía, la velocidad de operación, así como el costo y disponibilidad de los circuitos. En estos casos, el diseñador podría verse obligado a establecer un término medio entre la funcionalidad del dispositivo y las características físicas.

11.4 Ejemplos de diseño de PLD

Los siguientes ejemplos ilustran el diseño y la transformación de los diseños de circuitos secuenciales síncronos en dispositivos PLD.

EJEMPLO 11.1

Diseñar un circuito con una entrada x y una salida z que produzca una salida 1 siempre que las últimas cuatro entradas sean 1; es decir, un circuito que reconozca la secuencia de entrada $x = 1111$. Realizar el circuito en un dispositivo PAL16R6.

Utilizamos los procedimientos de diseño descritos en el capítulo 8 para obtener el diagrama de estados y la tabla reducida de estados de las figuras 11.25a y b. Elegimos la asignación de estados para este ejemplo de manera arbitraria, como sigue:

$$A = 00$$
$$B = 01$$
$$C = 10$$
$$D = 11$$

Las figuras 11.25c y d muestran la tabla de transición y el mapa de salida resultantes, y en ellas podemos ver que se necesitan dos *flip-flops* y una salida combinatoria para realizar el circuito.

El análisis del circuito PAL16R6 de la figura 11.11 muestra que disponemos de seis *flip-flops* D y dos salidas combinatorias. Utilicemos los *flip-flops* en las salidas O_7 y O_6 para las variables de estado y_1 y y_2 y la salida combinatoria I/O_8 para la salida z.

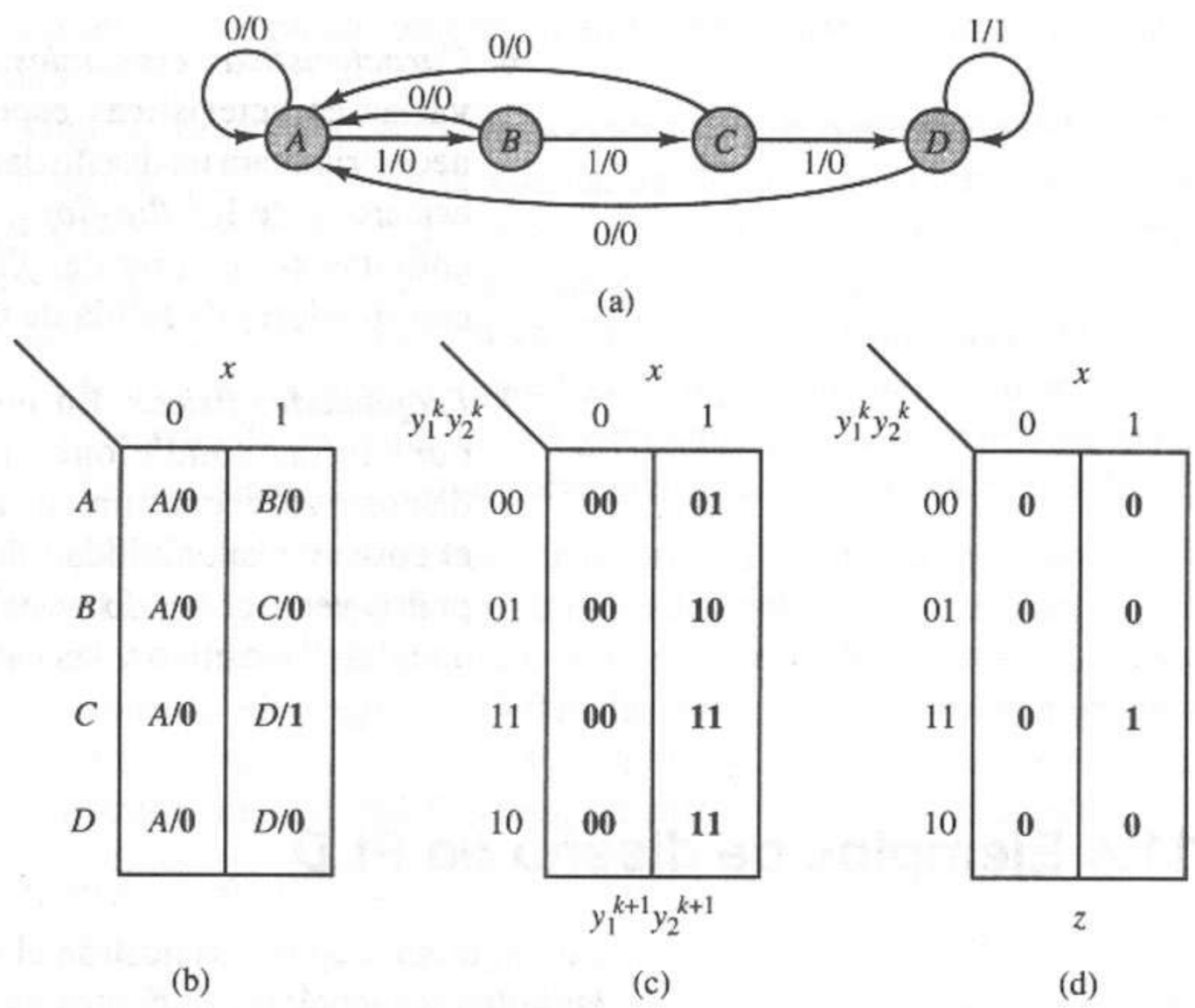


Figura 11.25 Circuito secuencial que detecta la secuencia de entrada x = 1111. **(a)** Diagrama de estados. **(b)** Tabla de estados. **(c)** Tabla de transición. **(d)** Mapa de salida.

Deducimos las ecuaciones de excitación para las entradas de excitación D_1 y D_2 del *flip-flop* D a partir de la tabla de transición de la figura 11.25c, y deducimos la ecuación de salida del mapa de salida de la figura 11.25d.

$$
\begin{aligned}
D_1 &= xy_1 + xy_2 \\
D_2 &= xy_1 + x\bar{y}_2 \\
z &= xy_1y_2 \\
&= \overline{\bar{x} + \bar{y}_1 + \bar{y}_2}
\end{aligned}
$$

Transformamos estas ecuaciones en el PAL16R6 de la figura 11.26. Para ilustrar los conceptos, sólo mostramos la parte del PAL16R6 utilizada en la implantación.

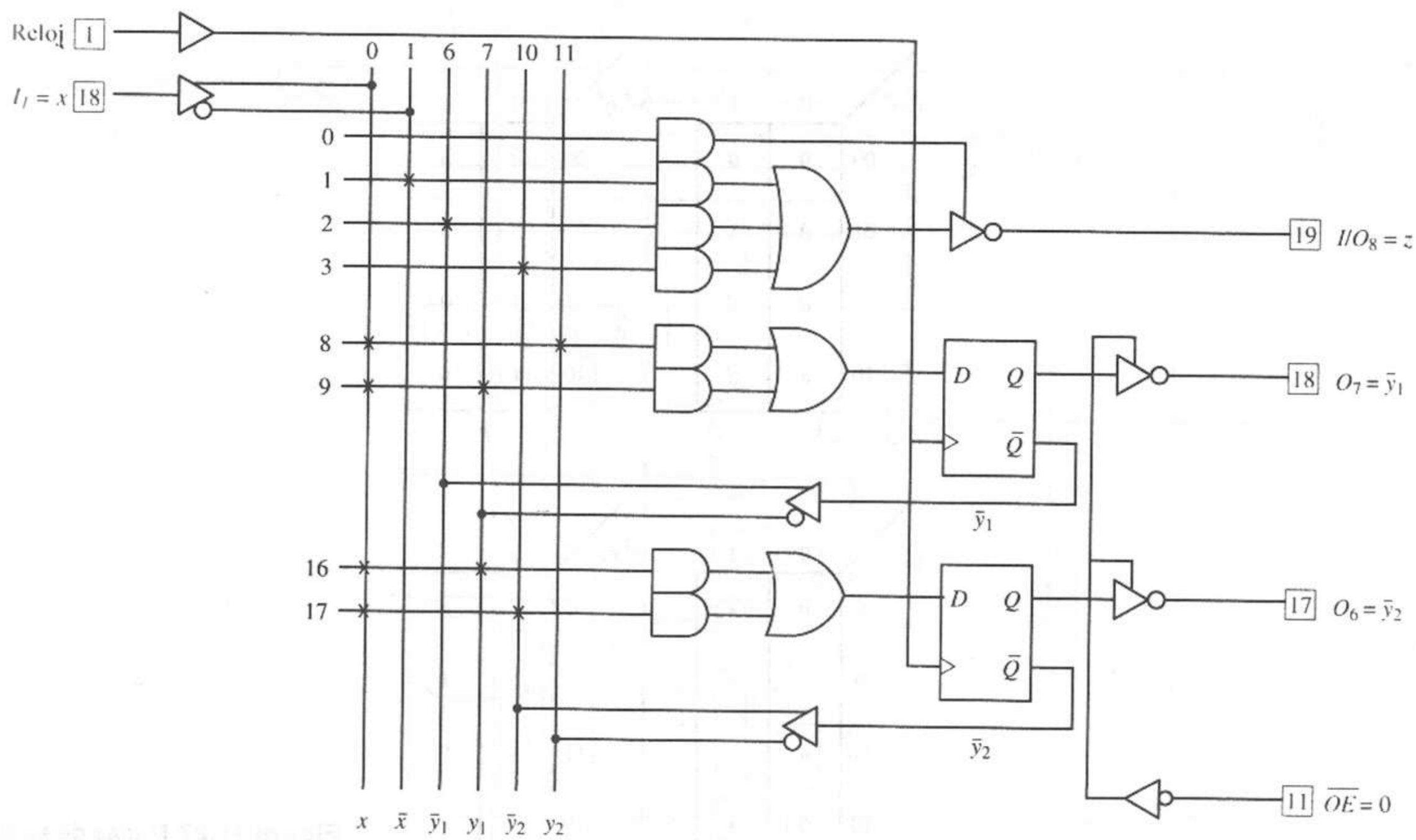


Figura 11.26 Realización PAL16R6 de un reconocedor de la secuencia $x = 1111$. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

EJEMPLO 11.2

Repetir el ejemplo 11.1 pero utilizando un dispositivo FPLS PLS155, con los elementos de almacenamiento configurados como *flip-flops* JK.

Al observar el diagrama del PLS155 en la figura 11.8 vemos que hay cuatro salidas con registro y ocho salidas combinatorias disponibles. Asociaremos las variables de estado y_1 y y_2 a las salidas con registro F_3 y F_2, respectivamente, y la salida z a la salida combinatoria B_7.

Comenzaremos deduciendo los mapas de excitación para los *flip-flops* JK a partir de la tabla de transición de la figura 11.25c. La figura 11.27 muestra estos mapas. A partir de los mapas de excitación, deducimos las siguientes ecuaciones de excitación:

$$J_1 = xy_2, \quad J_2 = x$$
$$K_1 = \bar{x}, \quad K_2 = \bar{x} + \bar{y}_1$$

Asociamos estas ecuaciones con el PLS155 como se muestra en la figura 11.28. Observe que M_2 y M_3 deben asumir el valor 1 para desactivar los *buffers* derivados de los *flip-flops*, lo cual configura los *flip-flops* para su funcionamiento JK.

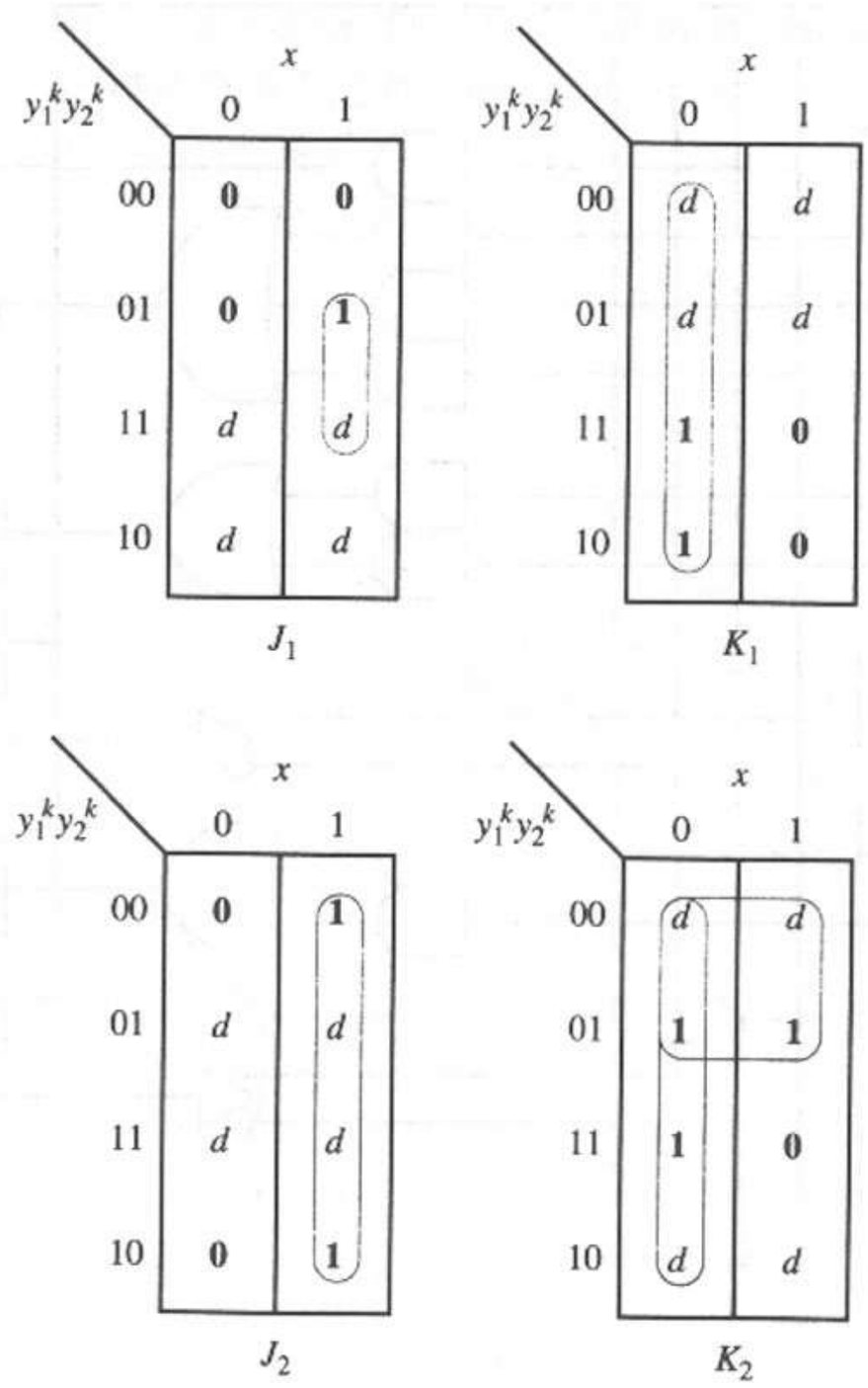


Figura 11.27 Mapas de excitación JK para el reconocedor de la secuencia $x = 1111$.

EMPLO 11.3

Diseñar un contador arriba/abajo binario de 4 bits con carga paralela. Las entradas incluyen las líneas de entrada de datos paralelos A_3 a A_0, la señal de control $L/\overline{C}$, que es alta para activar una carga paralela de datos y baja para activar el conteo; la señal de control $U/\overline{D}$, que es 1 para indicar conteo hacia arriba y 0 para indicar conteo hacia abajo, y un reloj CLK. Las salidas son la cuenta, con cuatro bits Q_3 a Q_0, y una señal OVR, que indica un desbordamiento; es decir, un conteo de 1111 a 0000 o de 0000 a 1111 en el siguiente pulso de reloj.

Dada esta descripción, el contador requiere seis entradas, cuatro entradas con registro para realizar el conteo, y una salida combinatoria para realizar la salida *OVR*. Al analizar la tabla 11.3, vemos que el contador se ajustaría a cualquiera de los PLD con registro de la tabla, con la excepción del PAL16R8, que no tiene salidas combinatorias. Los otros dispositivos tienen al menos cuatro salidas con registro y retroalimentación y al menos seis entradas. Para este ejemplo utilizaremos el dispositivo PAL16R4, con ocho entradas exclusivas, cuatro salidas con registro y cuatro salidas combinatorias.

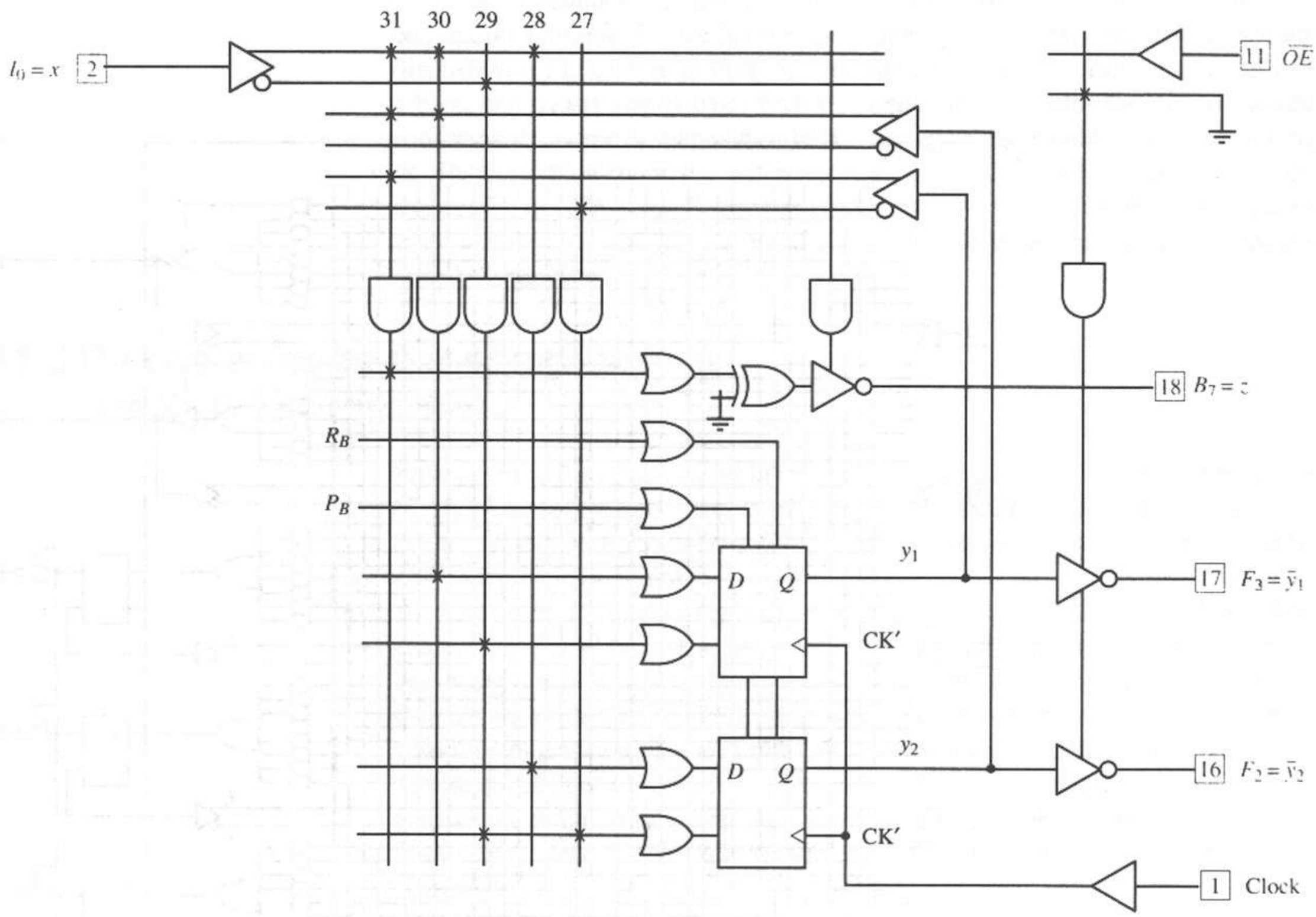


Figura 11.28 Realización PLS155 de un reconocedor de la secuencia $x = 1111$.

Como el PAL16R4 tiene *flip-flops* D, deducimos las siguientes ecuaciones de excitación siguiendo los procedimientos de diseño del capítulo 8. Los controladores de salida en las salidas con registros del PAL16R4 son inversores, al igual que en el PAL16R6 de la figura 11.11. Por tanto, hay que invertir los valores cargados en los *flip-flops*.

$$D_0 = \bar{A}_0(L/\bar{C}) + \bar{Q}_0(\overline{L/\bar{C}})$$
$$D_1 = \bar{A}_1(L/\bar{C}) + \bar{Q}_1(\bar{Q}_0(\overline{U/\bar{D}}) + Q_0(U/\bar{D})) \cdot (\overline{L/\bar{C}})$$
$$D_2 = \bar{A}_2(L/\bar{C}) + \bar{Q}_2(\bar{Q}_1\bar{Q}_0(\overline{U/\bar{D}}) + Q_1Q_0(U/\bar{D})) \cdot (\overline{L/\bar{C}})$$
$$D_3 = \bar{A}_3(L/\bar{C}) + \bar{Q}_3(\bar{Q}_2\bar{Q}_1\bar{Q}_0(\overline{U/\bar{D}}) + Q_2Q_1Q_0(U/\bar{D})) \cdot (\overline{L/\bar{C}})$$

Ocurre un desbordamiento cuando se cuenta hacia arriba desde $Q_3Q_2Q_1Q_0 = 1111$ o cuando se cuenta hacia abajo desde $Q_3Q_2Q_1Q_0 = 0000$. Por tanto, la expresión para la salida *OVR* es

$$OVR = Q_3Q_2Q_1Q_0(U/\bar{D}) + \bar{Q}_3\bar{Q}_2\bar{Q}_1\bar{Q}_0(\overline{U/\bar{D}})$$

Las ecuaciones de excitación y de salida se asocian con el PAL16R4 como se muestra en la figura 11.29.

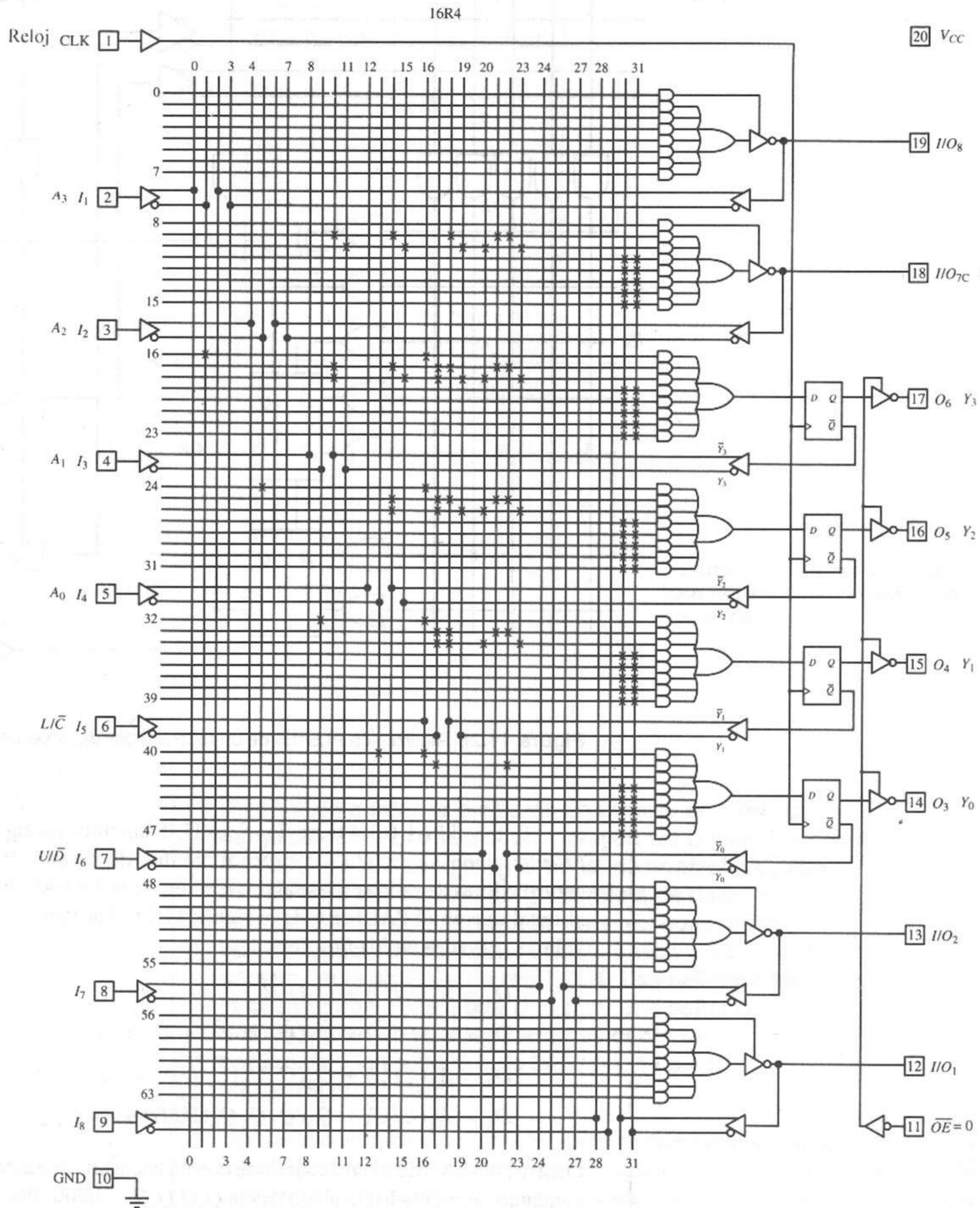


Figura 11.29 Realización PAL16R4 de un contador arriba/abajo binario de 4 bits con carga paralela. Copyright © Advanced Micro Devices, Inc., 1993. Reproducido con autorización del propietario de los derechos. Todos los derechos reservados.

Con frecuencia es recomendable utilizar *flip-flops* T en los diseños de contadores binarios. Si éste fuera el caso en el ejemplo anterior, elegiríamos un dispositivo FPLS, como el PLS155, que tiene tipos de *flip-flops* programables; o bien, uno de los dispositivos PAL que tienen macroceldas lógicas con salida programable, como el dispositivo EP910, en el que se puede programar el tipo de *flip-flops*. Realizaríamos entonces el diseño deduciendo las ecuaciones de excitación del *flip-flop* T, configurando cuatro salidas con registro para el funcionamiento del *flip-flop* T, y asociando después las ecuaciones de excitación con el dispositivo.

11.5 Diseño asistido por computadora de PLD secuenciales

En el capítulo 5 analizamos las herramientas de diseño asistido por computadora (CAD) que se utilizan en el diseño de PLD combinatorios. En esta sección examinamos varias características de estas herramientas que apoyan el diseño de un circuito secuencial.

Recordemos que la mayor parte de los paquetes de CAD para PLD permiten crear e introducir los diseños en varios formatos, incluidos diagramas esquemáticos, ecuaciones lógicas, tablas de verdad, diagramas de estado y tablas de estado. Como muestra la figura 11.30, cada diseño se traduce, o *compila*, en forma de ecuación lógica y después se minimizan las ecuaciones usando métodos similares a los analizados en el capítulo 3. Entonces, el diseño compilado se puede simular para ver si es correcto y estimar los parámetros de tiempo y otros. Cuando el diseño es correcto, las ecuaciones lógicas se transforman en el dispositivo PLD elegido.

La mayor parte de los paquetes de diseño de PLD utilizan un lenguaje de alto nivel para expresar los diseños en formato de ecuación lógica, tabla de verdad, tabla de estado o máquina de estado. Muchos de ellos también aceptan diseños creados mediante programas de captura de esquemas. En estos casos, el esquema se traduce a forma de ecuación lógica en el lenguaje utilizado por ese paquete. Por ejemplo, la figura 11.31 muestra un diagrama esquemático para un contador binario de 4 bits con una entrada *clear* asíncrona. Creamos este diagrama con el editor de esquemas *Design Architect* de Mentor Graphics y después utilizamos el programa *PLDesigner* de Minc para traducirlo al lenguaje PDL. El listado PDL resultante aparece en la figura 11.32 y las ecuaciones lógicas reducidas obtenidas por el compilador PDL aparecen en la figura 11.33.

La siguiente sección presenta un panorama de las características de apoyo de circuitos secuenciales del lenguaje PDL. Otros lenguajes de diseño de PLD son similares a PDL. El lector puede consultar los detalles en [9,10].

11.5.1 Representación del diseño de un circuito secuencial mediante PDL

Como vimos en el capítulo 5, PDL (PLDesigner Design Language) es típico de los lenguajes de alto nivel utilizados por las herramientas de diseño de PLD. Los diseños se pueden introducir en forma de ecuación, tabla de verdad, diagrama de estado, tabla de estado u otras formas de comportamiento. Como muestra el ejemplo de la figura 11.32, un archivo PDL incluye una sección de encabezado, la cual proporciona una descripción verbal del diseño, una sección opcional para

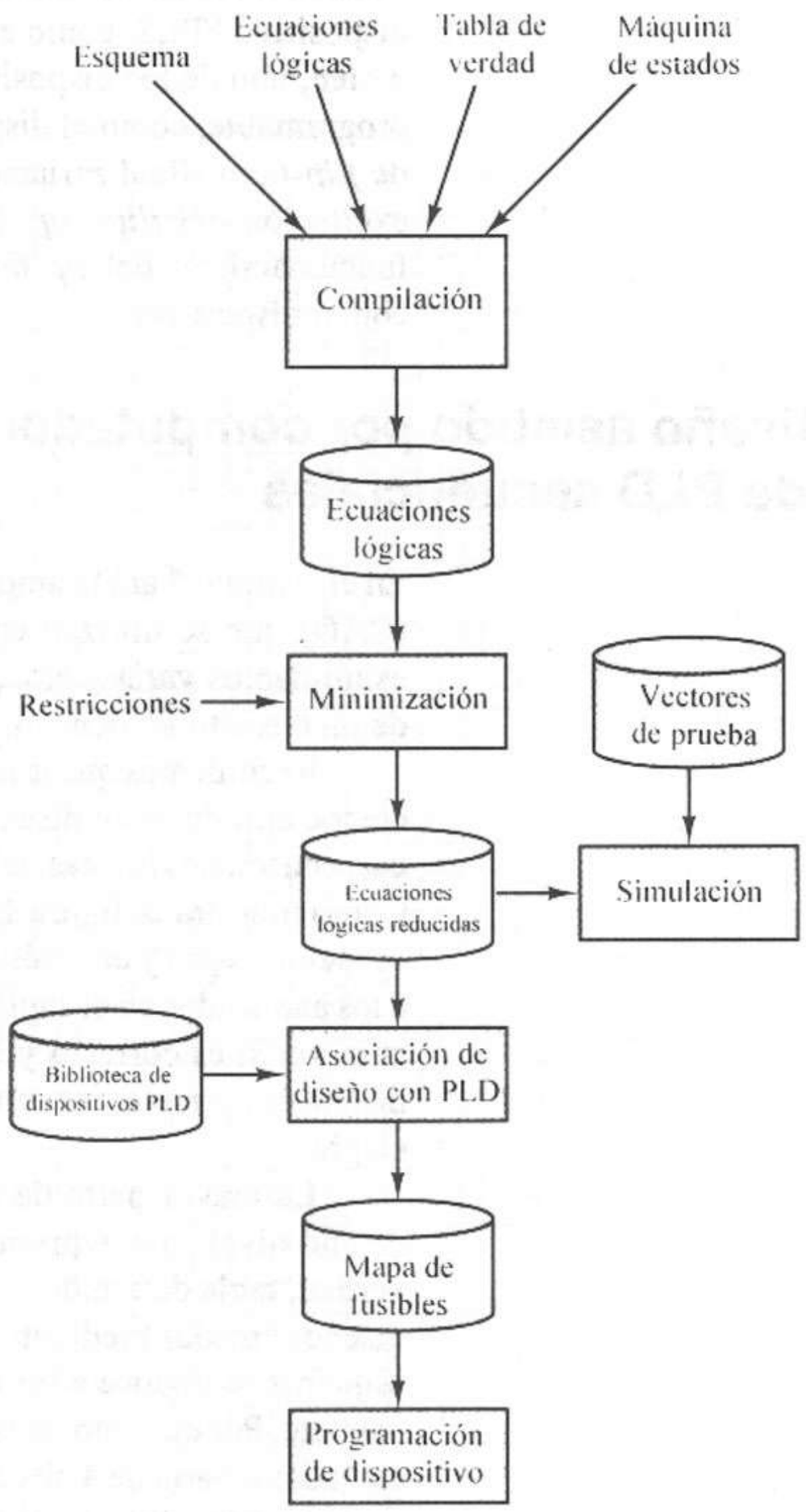


Figura 11.30 Proceso de diseño con PLD.

la definición de macros, la cual permite la representación simbólica de funciones y expresiones, y una sección de definición de funciones con las declaraciones de las señales de entrada y salida y la descripción de las ecuaciones lógicas, las tablas de verdad o la máquina de estados que describen las funciones por realizar.

Declaraciones de señales de entrada y salida

Todo diseño tiene varias entradas y salidas externas y, en algunos casos, líneas de entrada/salida bidireccionales. En un archivo PDL, estas señales se definen, o *declaran*, antes de enumerar la descripción funcional del diseño. En los dispositivos lógicos programables, las entradas externas pueden ser *pines* de

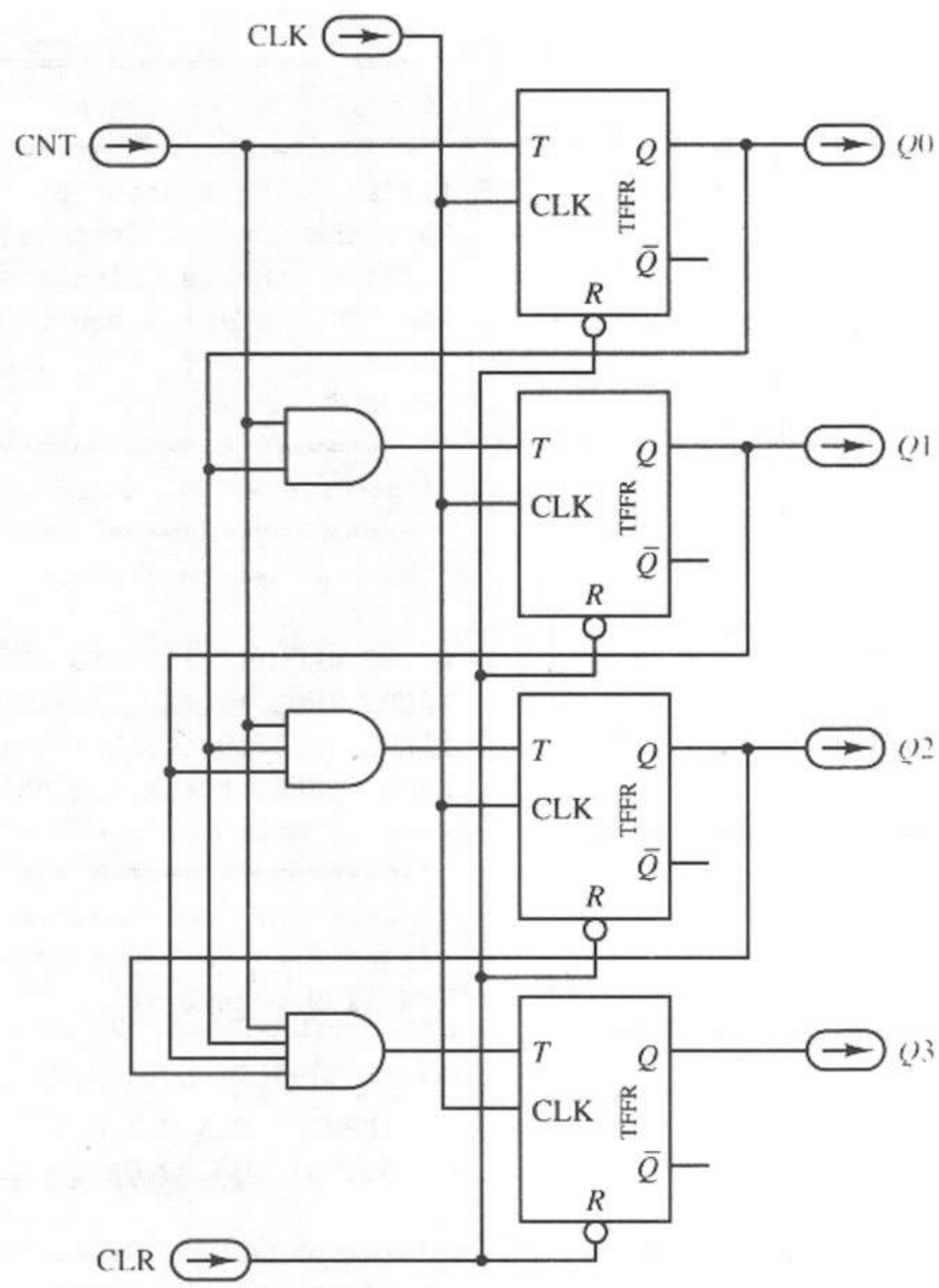


Figura 11.31 Diagrama esquemático del contador de cuatro bits.

entrada exclusiva o líneas de E/S cuyos controladores de salida están inhabilitados. Las salidas pueden ser combinatorias o con registros, con o sin un controlador de triple estado. En caso de tener registros, éste puede ser un *flip-flop* o *latch* con una entrada de reloj y posiblemente con entradas *reset* o actuales asíncronas. Los siguientes ejemplos ilustran varios formatos de declaración de una señal de entrada y salida.

```
INPUT x,y, [I3..I0];            ''entradas exclusivas:
OUTPUT x, [c3..c0]              ''salidas combinatorias
OUTPUT x,y ENABLED_BY oe;       '' salidas combinatorias con controladores de triple estado
BITPUT x1,x2 ENABLED_BY oe;     '' línea de E/S bidireccional
OUTPUT b0,b1 CLOCKED_BY clk HIDDEN;                          '' registro oculto
OUTPUT [q0..q3] CLOCKED_BY clk RESET_BY r PRESET_BY p;       '' salida con registro
```

La palabra reservada ENABLED_BY indica un controlador de triple estado asociado a una salida y define la señal de control para el controlador. La palabra

```
''=========================================
'' Sección de encabezado
''=========================================
TITLE      schematic.vpt ;
ENGINEER   Joe E. Student;
COMPANY    State University ;
PROJECT    EE401 Homework Project ;
REVISION   1.0 ;
COMMENT    Circuito contador hacia arriba de 4 bits;
''=========================================
'' Sección de definición de macros
''=========================================
''Macros para las compuertas AND y flip-flops T

MACRO AND2(i0,i1) (i0 * i1) ;
MACRO AND3(i0,i1,i2) (i0 * i1 * i2) ;
MACRO AND4(i0,i1,i2,i3) (i0 * i1 * i2 * i3) ;
MACRO TFFR(tt,clk,r,q,qb)      { q.t =  tt ; }

''=========================================
'' Sección de definición de funciones
''=========================================
FUNCTION schematic ;

'' Declara las entradas externas y las señales de salida
   INPUT   CLK,CLR,CNT ;
   OUTPUT  Q3,Q2,Q1,Q0 CLOCKED_BY CLK RESET_BY /CLR ;

'' Instancia tres compuertas AND
   MACRO  N$14  AND4(CNT,Q0,Q1,Q2) ;
   MACRO  N$13  AND3(CNT,Q0,Q1) ;
   MACRO  N$11  AND2(CNT,Q0) ;

'' Instancia cuatro flip-flops T
   TFFR(N$14,CLK,CLR,Q3,_x_x_x_x) ;
   TFFR(N$13,CLK,CLR,Q2,_x_x_x_x) ;
   TFFR(N$11,CLK,CLR,Q1,_x_x_x_x) ;
   TFFR(CNT,CLK,CLR,Q0,_x_x_x_x) ;

END schematic ;
```

Figura 11.32 Descripción en PDL del contador de cuatro bits, generada a partir del esquema.

reservada CLOCKED_BY indica la señal que controla una o más entradas de reloj de *flip-flops*, y de manera análoga, las palabras reservadas *RESET*_BY y *PRESET*_BY definen las señales que controlan las líneas de control *reset* y *preset* de *flip-flops*. Por último, la palabra reservada HIDDEN indica un registro que no controla un *pin* de salida; es decir, un registro oculto. Las palabras reservadas

```
Q3.CLK          = CLK ;   ''(1 término)
  .RESET        = /CLR ;  ''(1 término)
  .T            = Q2*Q1*Q0*CNT ;  ''(1 término)

Q2.CLK          = CLK ;   ''(1 término)
  .RESET        = /CLR ;  ''(1 término)
  .T            = Q1*Q0*CNT ;  ''(1 término)

Q1.CLK          = CLK ;   ''(1 término)
  .RESET        = /CLR ;  ''(1 término)
  .T            = Q0*CNT ;  ''(1 término)

Q0.CLK          = CLK ;   ''(1 término)
  .RESET        = /CLR ;  ''(1 término)
  .T            = CNT ;  ''(1 término)
```

Figura 11.33 Ecuaciones en PDL para el contador de cuatro bits, generadas por el compilador de PDL.

*RESET*_BY, *PRESET*_BY, CLOCKED_BY, HIDDEN y ENABLED_BY se utilizan u omiten según corresponda, para coincidir con las entradas y salidas reales de un dispositivo lógico en particular.

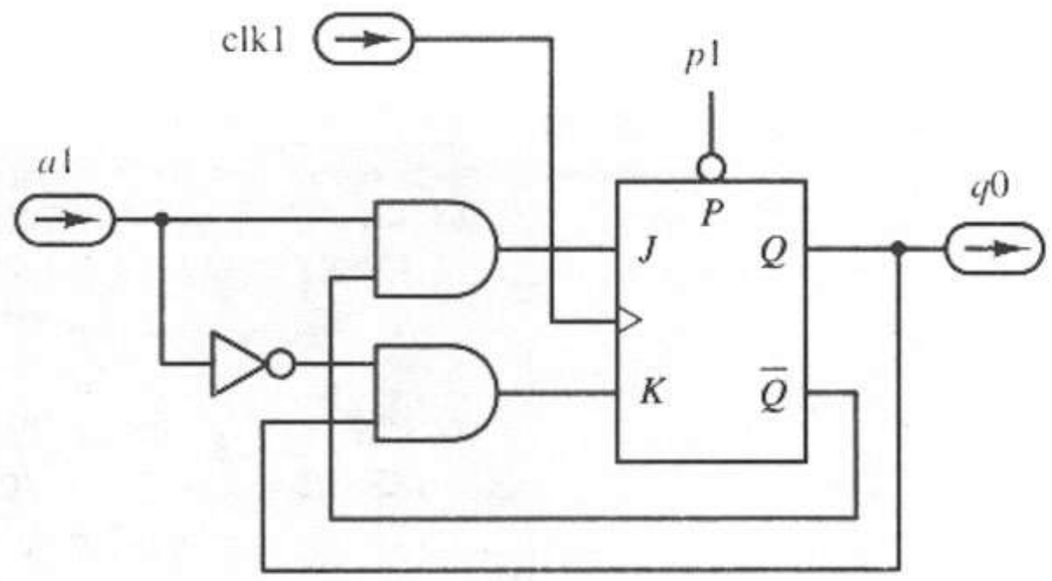
Ecuaciones lógicas

Las ecuaciones lógicas se expresan en PDL exactamente como se escribirían en una hoja de papel. En la tabla 5.5 enumeramos los operadores lógicos de PDL disponibles. Para las salidas con registro, cada salida de *flip-flop* tiene asignado un nombre en una declaración OUTPUT, y entonces todas las entradas de control y excitación del *flip-flop* se especifican agregando sufijos a este nombre. Por ejemplo, las figuras 11.34a y b muestran un circuito de *flip-flop* JK y su descripción en PDL. En este ejemplo, observe que las entradas J y K del *flip-flop* $q0$ se designan como $q0.J$ y $q0.K$, respectivamente. Observe además que no escribimos expresiones aparte para las entradas de reloj y *preset* del *flip-flop*, puesto que estas señales están definidas en la declaración OUTPUT de $q0$. Otros sufijos de entrada de excitación de *flip-flop* son {.D, .R, .S, .T}.

Descripción de máquina de estados

Recordemos que un diagrama de estados muestra los estados de una máquina secuencial, la transición entre estados provocada por cada combinación de entradas y las salidas producidas para cada combinación estado-entrada. En las máquinas Mealy, cada salida es una función del estado y de la entrada y, por tanto, se asigna a un arco de transición de estado. En las máquinas Moore, cada salida sólo es función del estado y, por tanto, se asigna al nodo que representa el estado.

Las figuras 11.35a y b presentan el diagrama de estados y la tabla de estados de una máquina Mealy con dos estados. La figura 11.35c muestra una descripción



(a)

```
INPUT     a1;
OUTPUT    q0 CLOCKED_BY clk1 PRESET_BY p1;
q0.J      = a1 * /q0;
q0.K      = /a1 * q0;
```

(b)

Figura 11.34 Circuito *flip-flop* JK. **(a)** Diagrama esquemático. **(b)** Descripción en PDL.

de máquina de estados de esta máquina. Como puede verse en el ejemplo, una descripción de máquina de estados en PDL incluye una declaración CLOCKED_BY, que define el reloj que dispara las transiciones de estado, y una declaración opcional STATE_BITS, que define los nombres de las variables de estado.

A cada estado se le asocia un nombre simbólico mediante una declaración STATE. La declaración de estado también define las transiciones de estado y las salidas asociadas al estado. Hay que definir dos tipos de información para cada estado: las transiciones de estado y las salidas. Las transiciones de estado se especifican mediante enunciados GOTO que indican el estado siguiente para cada combinación estado-entrada. Las salidas quedan definidas mediante ecuaciones lógicas.

Cuando hay varias combinaciones de entradas, se puede utilizar una construcción CASE GOTO, como en el ejemplo de las figuras 11.36a y b, el cual muestra una tabla de estados y la descripción en PDL de las transiciones de estado y las salidas para el estado *A*, para cada una de las cuatro combinaciones de las entradas x_1x_2.

Para definir las salidas de una máquina Moore se definen las expresiones de salida inmediatamente después del nombre del estado, como en la descripción parcial del contador binario de la figura 11.37. Si se desea utilizar una asignación de estados en particular, ésta se especifica con la construcción STATE_BITS. Primero se definen las variables de estado, seguidas por la asignación deseada para cada estado.

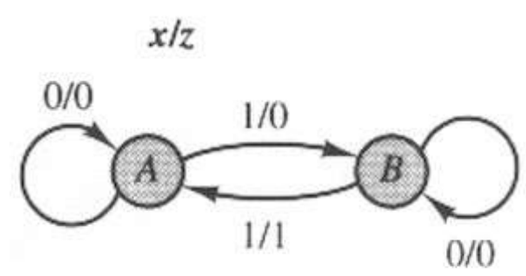
Formato de tabla de estados

Las máquinas de estados también se pueden expresar en forma de tabla de estados, mediante la construcción TRUTH_TABLE descrita anteriormente. En este caso,

Estado actual \ Entrada x^k	0	1
A	A/0	B/0
B	B/0	A/1

Estado siguiente/salida

(a)



(b)

```
STATE_MACHINE example;
    CLOCKED_BY clk;                 " transiciones de estado para disparar el reloj
    STATE_BITS [y];                 " define la variable de estado
    STATE state_A [0];              " define acciones si está en estado A
        IF (x) THEN
            BEGIN                   " acción para x = 1
                z = 0;              " la salida es 0
                GOTO state_B;       " el estado siguiente es B
            END;
        ELSE
            BEGIN                   " acción para x = 0
                z = 0;
                GOTO state_A;
            END;
    STATE state_B [1];              " define acciones si está en estado B
        IF (x) THEN
            BEGIN                   " acción para x = 1
                z = 1;
                GOTO state_A;
            END;
        ELSE
            BEGIN                   " acción para x = 0
                z = 0;
                GOTO state_B;
            END;
END example;
```

(c)

Figura 11.35 Representación de una máquina de estados en PDL. **(a)** Tabla de estados. **(b)** Diagrama de estados. **(c)** Descripción en PDL.

cada variable de estado aparece como una entrada y una salida en la tabla de verdad, de modo que la salida define el estado siguiente de esa variable. Las figuras 11.38a y b muestran las descripciones en PDL de la tabla de estados de la figura 11.36a.

11.5.2 Procesamiento de un archivo de diseño en PDL

Después de preparar una descripción en PDL de un diseño, se llama al compilador de PDL para traducir y reducir el diseño. Esto implica varios pasos. En el caso de una descripción de comportamiento, como una tabla de estados o alguna otra

$y_1 y_2$ \ $x_1 x_2$	00	01	11	10
00 = *A*	*A*, 0	*A*, 0	*B*, 0	*C*, 0
01 = *B*	*C*, 0	*B*, 0	*D*, 1	*A*, 1
11 = *C*	*A*, 1	*C*, 0	*C*, 1	*D*, 0
10 = *D*	*A*, 0	*B*, 1	*D*, 1	*D*, 1

(a)

```
STATE A: (/x1 * /x2):                    "acción para x1=0, x2=0
        BEGIN
            z = 0;
            GOTO A;
        END;
    (/x1 * x2):                          "acción para x1=0, x2=1
        BEGIN
            z = 0;
            GOTO A;
        END;
    (x1 * x2):                           "acción para x1=1, x2=1
        BEGIN
            z = 0;
            GOTO B;
        END;
    (x1 * /x2):                          "acción para x1=1, x2=0
        BEGIN
            z = 0;
            GOTO C;
        END;
END;
```

(b)

Figura 11.36 Uso de una construcción CASE para definir las transiciones de estado. **(a)** Tabla de estados. **(b)** Descripción PDL parcial, mostrando las acciones para el estado A.

descripción de máquina de estados, el compilador sintetiza primero la máquina, convirtiendo la descripción de máquina de estados en ecuaciones lógicas para todas las salidas y entradas de excitación de *flip-flop*. Una vez que el diseño está en forma de ecuación lógica, se simplifican las ecuaciones a su forma SOP de dos niveles, que se puede transformar en un arreglo AND/OR PAL o PLA.

En el proceso de simplificación de las ecuaciones, éstas se minimizan mediante una de cuatro opciones que puede especificar el usuario. La primera consiste en no realizar reducción alguna, sino dejar las ecuaciones en forma SOP. La segunda opción consiste en aplicar el algoritmo *Espresso*, el cual reduce las ecuaciones con rapidez y con poco uso de memoria, aunque no necesariamente produce una solución óptima. La tercera opción utiliza el algoritmo *Espresso*

```
OUTPUTS c3,c2,c1,c0;
STATE_MACHINE count4;
    CLOCKED_BY clk;
    STATE_BITS [q3..q0];
    STATE cnt0 [0000b];
        c3 = 0; c2 = 0; c1 = 0; c0 = 0; "Salidas Moore
        if (clr) THEN
            GOTO cnt0;
        ELSE
            IF (cnt) THEN
                GOTO cnt1;
            ELSE
                GOTO cnt0;

    STATE cnt1 [0001b];
        c3 = 0; c2 = 0; c1 = 0; c0 = 1; "Salidas Moore
        if (clr) THEN
            GOTO cnt0;
        ELSE
            IF (cnt) THEN
                GOTO cnt2;
            ELSE
                GOTO cnt1;
    ....

END count4;
```

Figura 11.37 Descripción de una máquina de estados para un contador de 4 bits con salidas Moore.

con alguna de las técnicas *Quine-McCluskey* para deducir una mejor cubierta. La cuarta opción utiliza el método *Quine-McCluskey* completo, el cual produce una solución óptima pero a costa de un mayor tiempo de cómputo y más uso de memoria.

Por ejemplo, a partir de las tablas de estado de la figura 11.38 el compilador de PDL generó las ecuaciones lógicas dadas en la figura 11.39.

Después de compilar un diseño, el siguiente paso consiste en verificar que sea correcto, mediante una simulación funcional. El lenguaje permite especificar vectores de prueba y controles de simulación dentro del archivo de diseño, de modo que la simulación se efectúe inmediatamente después de la compilación utilizando la herramienta *PLDsim* del sistema PLDsynthesis. Cuando PLDsynthesis se integra en otro ambiente de diseño, como *Falcon Framework* de Mentor Graphics, se pueden utilizar otros simuladores, como el simulador lógico *QuickSum II* de Mentor Graphics. El lector puede consultar [9] si desea más detalles acerca de la simulación dentro de los ambientes PLDesigner y Falcon Framework.

```
FUNCTION statemach2;
MACRO StateA 0,0;
MACRO StateB 0,1;
MACRO StateC 1,1;
MACRO StateD 1,0;
INPUT x1, x2, clk;
OUTPUT z;
OUTPUT y1, y2 CLOCKED_BY clk;
TRUTH_TABLE
y1, y2,   x1,   x2   ::   y1, y2,   z;
StateA,   0,    0    ::   StateA,   0;
StateA,   0,    1    ::   StateA,   0;
StateA,   1,    0    ::   StateC,   0;
StateA,   1,    1    ::   StateB,   0;
StateB,   0,    0    ::   StateC,   0;
StateB,   0,    1    ::   StateB,   0;
StateB,   1,    0    ::   StateA,   1;
StateB,   1,    1    ::   StateD,   1;
StateC,   0,    0    ::   StateA,   1;
StateC,   0,    1    ::   StateC,   0;
StateC,   1,    0    ::   StateD,   0;
StateC,   1,    1    ::   StateC,   1;
StateD,   0,    0    ::   StateA,   0;
StateD,   0,    1    ::   StateB,   1;
StateD,   1,    0    ::   StateD,   1;
StateD,   1,    1    ::   StateD,   1;
END;
END statemach2;
```

(a)

```
FUNCTION statemach;
INPUT x1,x2,clk;
OUTPUT z;
OUTPUT y1,y2 CLOCKED_BY clk;
TRUTH_TABLE
y1,   y2,   x1,   x2   ::   y1,   y2,   z;
0,    0,    0,    0    ::   0,    0,    0;
0,    0,    0,    1    ::   0,    0,    0;
0,    0,    1,    1    ::   0,    1,    0;
0,    0,    1,    0    ::   1,    1,    0;
0,    1,    0,    0    ::   1,    1,    0;
0,    1,    0,    1    ::   0,    1,    0;
0,    1,    1,    1    ::   1,    0,    1;
0,    1,    1,    0    ::   0,    0,    1;
1,    1,    0,    0    ::   0,    0,    1;
1,    1,    0,    1    ::   1,    1,    0;
1,    1,    1,    1    ::   1,    1,    1;
1,    1,    1,    0    ::   1,    0,    0;
1,    0,    0,    0    ::   0,    0,    0;
1,    0,    0,    1    ::   0,    1,    1;
1,    0,    1,    1    ::   1,    0,    1;
1,    0,    1,    0    ::   1,    0,    1;
END;
END statemach;
```

(b)

Figura 11.38 Descripciones en PDL de una tabla de estados. **(a)** Nombres de estado simbólicos utilizados. **(b)** Valores dados a las variables de estado.

El siguiente paso del proceso consiste en asociar las ecuaciones reducidas con el dispositivo elegido. El sistema PLDesigner incluye una biblioteca de dispositivos de la cual pueden elegirse los dispositivos más adecuados para un diseño que satisfagan los criterios establecidos por el usuario.

En PLDsynthesis, estos criterios dados por el usuario, o valores de restricción, incluyen el tipo de paquete, la familia lógica, el fabricante, el intervalo de temperatura, la corriente máxima, la frecuencia máxima, el retardo máximo y el precio de los componentes. Cada restricción tiene asociado un factor de peso, de modo que se pueda elegir el dispositivo dando más importancia a los valores de restricción que el diseñador considere más importantes.

La salida de la operación de ajuste del dispositivo es un mapa de fusibles, que se puede cargar en un programador de dispositivos para programar el circuito. En ciertos casos, se puede proporcionar información de simulación al programa-

```
Z.EQN     = X1*X2*Y2
          + Y1*/X1*/X2*Y2
          + /Y1*X1*Y2
          + Y1*X2*/Y2
          + Y1*X1*/Y2 ; "(5 términos)

Y1.CLK    = CLK ; "(1 term)
.D        = Y2*/Y1*/X2*/X1
          + Y1*X1
          + /Y2*/X2*X1
          + Y2*Y1*X2
          + Y2*X2*X1 ; "(5 términos)

Y2.CLK    = CLK ; "(1 term)
.D        = Y1*/X1*X2
          + Y1*Y2*X2
          + /Y1*/Y2*X1
          + /Y1*Y2*/X1 ; "(4 términos)
```

Figura 11.39 Ecuaciones reducidas de excitación y salida para la tabla de estados de la figura 11.38.

dor del dispositivo para que pueda probarlo y comparar la operación real con los resultados simulados.

11.6 Resumen

En este capítulo examinamos los dispositivos lógicos programables, como los PLD y los FPGA, que se pueden utilizar para realizar circuitos secuenciales síncronos y asíncronos. Ilustramos la asociación de las máquinas Mealy y Moore con estos dispositivos. Por último, examinamos las herramientas de CAD que pueden servir para desarrollar diseños basados en PLD, incluyendo una descripción del sistema PLDsynthesis de Minc.

REFERENCIAS BIBLIOGRÁFICAS

1. PHILIPS, *Programmable Logic Devices (PLD) Data Handbook*. Sunnyvale, CA: Philips Semiconductors, 1994.
2. ADVANCED MICRO DEVICES, *PAL Device Data Book*. Sunnyvale, CA: Advanced Micro Devices, Inc., 1990.
3. ALTERA, *User-configurable Logic Databook*. Santa Clara, CA: Altera Corp., 1988.
4. XILINX, *The Programmable Gate Array Design Handbook*. San Jose, CA: Xilinx, Inc., 1987.
5. ACTEL, *ACT Family Field Programmable Gate Array Databook*. Sunnyvale, CA: Actel Corp., 1991.
6. PARAG K. LALA, *Digital System Design Using Programmable Logic Devices*. Englewood Cliffs, NJ: Prentice-Hall, 1990.
7. MARTIN BOLTON, *Digital Systems Design with Programmable Logic*. Reading, MA: Addison-Wesley, 1990.

8. MINC, INC., *PLDesigner Student Version Manual*. New York: McGraw-Hill Publishing Co., 1990.

9. *PLDSYNTHESIS USER'S MANUAL*, Wilsonville, OR: Mentor Graphics Corp.

10. *PDL LANGUAGE REFERENCE MANUAL*, Wilsonville, OR: Mentor Graphics Corp.

PROBLEMAS

11.1 Diseñe una implantación PAL16R4 de un circuito secuencial síncrono que reconozca la secuencia de entrada 1010. Las secuencias se pueden traslapar. Por ejemplo,

$$x = 001010010101011 10$$
$$z = 00000100001010000$$

Deduzca las ecuaciones lógicas y después trace el diagrama de circuito PAL16R4 usando el formato de la figura 11.26.

11.2 Diseñe el circuito secuencial de cuatro estados definido por la siguiente tabla de estados, utilizando la asignación de estados indicada. Identifique el número de *pines* por asignar a cada entrada, salida y variable de estado; escriba las ecuaciones lógicas en un formato adecuado para su implantación en:

(a) Un PAL16R6.

(b) Un PLS155, con los *flip-flops* configurados para funcionamiento JK.

y_1	y_2
0	0
0	1
1	1
1	0

	x = 0	x = 1
A	*B*/0	*C*/0
B	*D*/0	*A*/1
C	*A*/1	*D*/0
D	*D*/1	*B*/1

11.3 Para el circuito descrito por la tabla de estados y asignación de estados dadas abajo, determine una implantación PLS155, configurando los *flip-flops* para funcionamiento JK.

(a) Escriba las ecuaciones lógicas e indique el número de *pines* del PLS155 por asignar a cada entrada, salida y variable de estado.

(b) Trace el diagrama lógico con el formato de la figura 11.28.

y_1	y_2
0	0
0	1
1	1
1	0

	x = 0	x = 1
A	*B*/0	*D*/0
B	*C*/0	*A*/0
C	*D*/0	*B*/0
D	*A*/1	*C*/1

11.4 Repita el problema 11.3, pero implante el circuito con un 22V10, indicando la configuración de cada macrocelda utilizada en el diseño.

11.5 Dadas las siguientes tabla de estados reducida y asignación de estados, determine las ecuaciones lógicas del circuito secuencial síncrono y trace el diagrama lógico, utilizando los dispositivos siguientes.

(a) PAL16R6.

(b) PLS155, con los *flip-flops* configurados para funcionamiento JK.

(c) 22V10.

y_1	y_2		x = 0	x = 1
0	0	A	*A*/0	*B*/0
0	1	B	*C*/0	*B*/0
1	1	C	*D*/0	*B*/0
1	0	D	*A*/1	*B*/0

11.6 Diseñe el circuito secuencial de modelo Moore descrito por la tabla de estados y asignación de estados dadas a continuación, utilizando los dispositivos siguientes (no utilice salidas combinatorias). Deduzca las ecuaciones lógicas y trace después el diagrama lógico.
(a) Utilice un PLS155 con los *flip-flops* configuradas para funcionamiento JK.
(b) Utilice un 22V10.

y_1	y_2		x = 0	x = 1	z
0	0	*A*	*A*	*B*	1
0	1	*B*	*C*	*B*	0
1	1	*C*	*D*	*B*	0
1	0	*D*	*B*	*A*	1

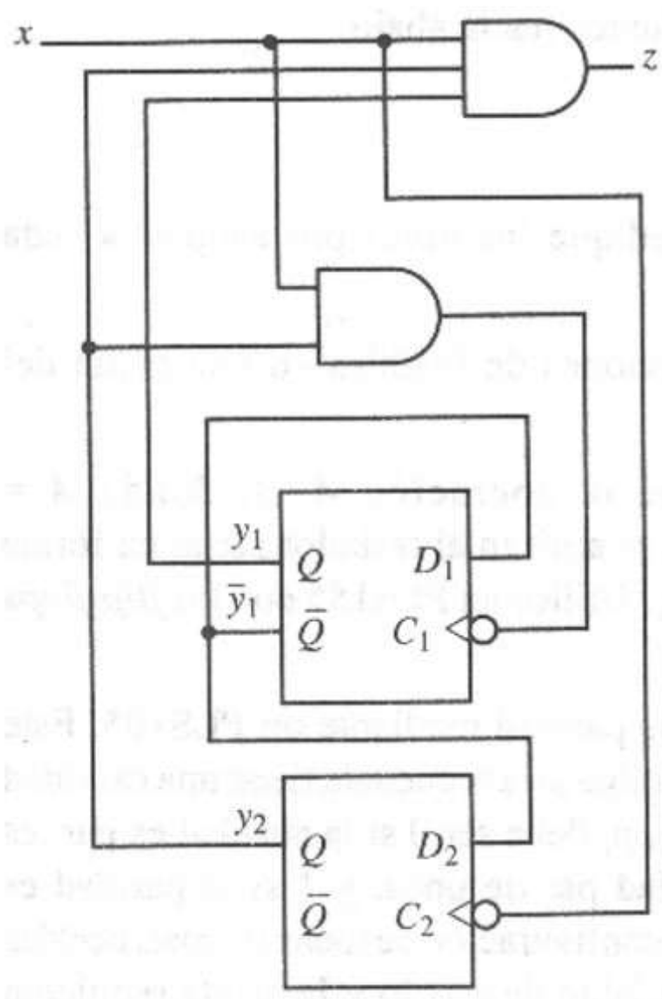


Figura P11.8

11.7 Determine una realización PAL16R6 para la siguiente tabla de estados reducida, utilizando la asignación de estados *one-hot* dada.

y_1	y_2	y_3	y_4	y_5	y_6		x = 0	x = 1
1	0	0	0	0	0	*A*	*B*/0	*D*/0
0	1	0	0	0	0	*B*	*A*/0	*C*/1
0	0	1	0	0	0	*C*	*D*/1	*C*/0
0	0	0	1	0	0	*D*	*B*/1	*E*/1
0	0	0	0	1	0	*E*	*C*/0	*A*/0
0	0	0	0	0	1	*F*	*E*/0	*F*/1

11.8 Implante el circuito secuencial asíncrono con pulsos dado en la figura P11.8, utilizando un Altera EP910. Trace el diagrama lógico en el formato de la figura 11.26, con las macroceldas dispuestas como en la figura 11.17. (*Sugerencia*: se necesitan tres macroceldas.)

11.9 Diseñe un registro de carga paralela de 8 bits con un PLS105. El registro debe tener 8 entradas dato D_7–D_0, 8 salidas Q_7–Q_0, la entrada de reloj *CLK*, y la entrada de control *preset* PRE. Trace el diagrama lógico con el formato de la figura 11.28 o marque las entradas, salidas y conexiones de fusibles en una copia del diagrama PLS105.

11.10 Diseñe un registro de corrimiento bidireccional de 4 bits e implántelo en un PAL16R6. El registro de corrimiento debe tener las entradas seriales *Sin-Right* y *Sin-Left*, las entradas paralelas *A, B, C, D*, las salidas paralelas $Q_AQ_BQ_CQ_D$, una entrada de reloj *CLK*, y dos entradas de selección de función S_1S_0. En la tabla siguiente definimos las funciones del registro de corrimiento:

S_1	S_0	Función
0	0	No opera
0	1	Carga
1	0	Corrimiento a la derecha
1	1	Corrimiento a la izquierda

(a) Deduzca las ecuaciones lógicas e indique los *pines* por asignar a cada entrada, salida y variable de estado. Observe que los controladores de salida del PAL16R6 son inversores.

(b) Trace el diagrama lógico en el formato de la figura 11.26, o bien marque las entradas, salidas y conexiones de fusibles en una copia del diagrama del PAL16R6.

11.11 Diseñe un contador módulo 12, arriba/abajo, de 4 bits, mediante un PLS155, con los *flip-flops* configurados para funcionamiento JK. El contador debe tener entradas paralelas $\{D,C,B,A\}$, salidas $\{Q_D, Q_C, Q_B, Q_A\}$, entrada de reloj *CLK* y dos entradas para selección de función S_1S_0. Las funciones del contador están definidas en la tabla siguiente:

S_1	S_0	Modo
0	0	No opera
0	1	Carga
1	0	Conteo hacia arriba
1	1	Conteo hacia abajo

(a) Deduzca las ecuaciones lógicas e indique los *pines* por asignar a cada entrada, salida y variable de estado.

(b) Marque las entradas, salidas y conexiones de fusibles en una copia del diagrama del PLS155.

11.12 Diseñe un restador serial que realice la operación $A-B$, donde $A = a_{n-1}...a_1a_0$ y $B = b_{n-1}...b_1b_0$. Los operandos se aplican al restador serial en forma secuencial, comenzando con los bits a_0 y b_0. Utilice un PLS155 con los *flip-flops* configurados para funcionamiento JK.

11.13 Diseñe un circuito serial para detección de paridad mediante un PLS105. Este circuito recibe una secuencia de bits y determina si la secuencia tiene una cantidad par o impar de unos. La salida del circuito, *p*, debe ser 0 si la paridad es par; es decir, si la secuencia contiene una cantidad par de unos, y 1 si la paridad es impar. Trace el diagrama lógico e indique la configuración de todas las macroceldas utilizadas en el diseño. Observe que la variable de estado y la salida requieren *flip-flops* aparte en el PLS105.

11.14 Diseñe un circuito lógico para implantar la unidad de control de la máquina de dulces diseñada en el ejemplo 8.18. Utilice una asignación de estados *one-hot*, e implante el circuito en un PAL16R6. Trace el diagrama lógico o marque las entradas, salidas y conexiones de fusibles en una copia del diagrama del PAL16R6.

11.15 Diseñe un circuito lógico para la unidad de control del multiplicador binario cuyo diagrama ASM fue diseñado en el ejemplo 8.20, mediante un 22V10 y una asignación de estados *one-hot*.

11.16 Diseñe un contador binario de un bit que utilice un único CLB de la familia XC2000 de Xilinx. El contador debe tener una entrada de reloj, *Clock*, una entrada de habilitación, *Enable*, una entrada *reset* síncrona, *Reset*, y una única salida, *Q*. Trace el diagrama CLB, etiquetando las entradas y la salida y resaltando las trayectorias a través de los multiplexores. Además, elabore la tabla de verdad del bloque de lógica combinatoria.

11.17 Repita el diseño del circuito secuencial del problema 11.2, pero implante el circuito con dos CLB de la familia XC2000 de Xilinx. Trace los diagramas lógicos de los CLB, mostrando la configuración de los multiplexores; indique

también las interconexiones entre los CLB, la entrada x y la salida z. Además, enumere la tabla de verdad de cada bloque lógico combinatorio CLB.

11.18 Diseñe el circuito restador serial descrito en el problema 11.12, utilizando un único CLB XC4000 para realizar el circuito. Muestre la configuración de cada multiplexor del CLB, y proporcione las tablas de verdad o listas de mintérminos de los bloques de lógica combinatoria utilizados.

11.19 Realice cada una de las siguientes funciones de conmutación con un único módulo lógico de la familia ACT-1 de Actel.

(a) $f(a, b, c) = a\bar{b}\bar{c}$.

(b) $f(a, b, c, d) = \bar{a}b + ac + d$.

(c) Un multiplexor 4 a 1.

(d) $f(a, b, c) = ab + ac + bc$.

11.20 Implante los siguientes dispositivos de memoria con módulos lógicos de la familia ACT-1 de Actel. Trace el diagrama lógico de cada uno.

(a) Un *latch* D con compuertas, con entradas D y C, y una salida Q (utilice un módulo).

(b) Un *flip-flop* D maestro-esclavo, con entradas D y C, y una salida Q, de modo que la salida D cambie en la transición positiva de C (utilice dos módulos).

11.21 Diseñe el circuito para detección de paridad descrito en el problema 11.13 mediante módulos lógicos de la familia ACT-1. Utilice el diseño de *flip-flop* D maestro-esclavo del problema 11.20b como elemento de memoria. Trace el diagrama lógico, mostrando todas las conexiones con los módulos lógicos.

11.22 Escriba un modelo en PDL del circuito secuencial síncrono representado por las siguientes tabla y asignación de estados:

(a) utilizando el formato de tabla de estados.

(b) utilizando el formato de máquina de estados.

y_1	y_2
0	0
0	1
1	1
1	0

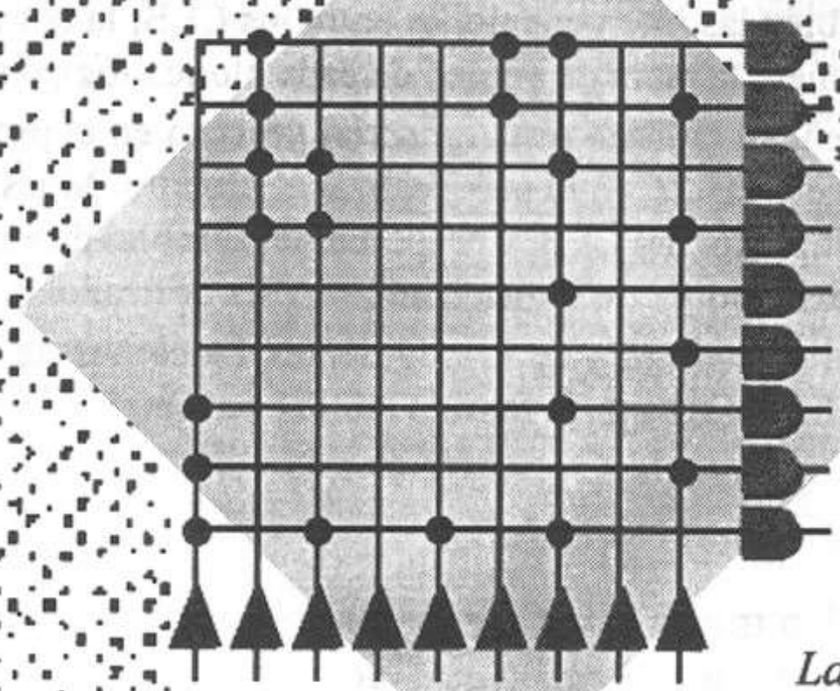
	x = 0	x = 1
A	$A/0$	$B/0$
B	$C/0$	$B/0$
C	$D/0$	$B/0$
D	$B/1$	$A/0$

11.23 Escriba un modelo de máquina de estados en PDL para el reconocedor de secuencias descrito en el problema 11.1.

11.24 Deduzca las ecuaciones de excitación y de salida para el circuito secuencial síncrono descrito en el problema 11.4; a continuación, escriba un modelo en PDL del circuito, en términos de estas ecuaciones. Suponga que el circuito debe implantarse en un PAL16R4.

11.25 Diseñe un modelo de máquina de estados en PDL del contador arriba/abajo módulo 12 descrito en el problema 11.11.

12



La verificación es una parte crítica del proceso de fabricación de un dispositivo, tarjeta de circuitos o sistema con circuitos digitales. Cada parte que sale de una fábrica debe verificarse cabalmente para comprobar que funcione como se pretende. El esfuerzo necesario para diseñar un procedimiento de verificación y la propia verificación de un circuito pueden representar un aumento significativo en su costo total y tiempo de entrega. En consecuencia, tanto el desarrollo de una estrategia para la verificación de fallas, o prueba, como la aplicación de la prueba a cada circuito deben realizarse de la manera más eficiente posible, a la vez que deben proporcionar una alta probabilidad de detección de un circuito defectuoso.

Este capítulo presentará los tipos de fallas que ocurren en los circuitos lógicos digitales. Después describiremos algunos métodos para deducir pruebas que detecten y localicen fallas en los circuitos lógicos secuenciales y combinatorios. Para facilitar el proceso de verificación, presentaremos métodos de diseño que mejoran la verificabilidad de los circuitos lógicos. Por último, describiremos el diseño de las características de verificación integradas que se pueden utilizar en los circuitos integrados digitales y en las tarjetas de circuito.

Verificación de circuitos lógicos y diseño verificable

12.1 Verificación de circuitos lógicos digitales

Los dos objetivos fundamentales de la verificación son la detección y la localización de fallas. Definimos de manera informal una *falla* como cualquier condición que causa el funcionamiento inadecuado de un dispositivo. La *verificación para la detección de fallas* es el proceso que determina si existe o no una falla en un dispositivo dado. Un conjunto de entradas a un circuito lógico que puede servir para detectar una falla en el circuito es un *conjunto de pruebas para detección de fallas* (FDTS). La *verificación para la localización de fallas* es el proceso que determina en qué consiste la falla presente en un dispositivo con fallas. Un *conjunto de prueba para la localización de fallas* (FLTS) es un conjunto de entradas que pueden servir para localizar una falla. La *cobertura de fallas* de una verificación para la detección o localización de fallas es el porcentaje de todas las fallas potenciales en un circuito detectadas o localizadas por la prueba.

La figura 12.1 muestra una configuración típica de prueba de un circuito digital. El *circuito bajo prueba* (CUT) se estimula al aplicar los vectores de prueba de un FDTS o FLTS a las entradas del CUT. Evaluamos la operación del CUT capturando sus respuestas a los vectores de prueba y comparándolas con los valores esperados. En la mayor parte del equipo de verificación automática (ATE), como muestra la figura 12.1a, los vectores de prueba se almacenan en una memoria, de donde son recuperados y aplicados al CUT por medio de un microprocesador. En algunos sistemas ATE y en circuitos con autoverificación integrada (BIST), los patrones de prueba se generan en forma automática mediante circuitos especiales, como en la figura 12.1b.

Las respuestas del CUT a los vectores de prueba aplicados se evalúan al capturarlas y compararlas con los valores esperados. En la mayor parte de los sistemas ATE, esto se realiza recuperando los valores esperados de una memoria cada vez que se aplica un vector de prueba, y comparando cada uno con la salida correspondiente del circuito. En la mayor parte de los casos, las listas de valores esperados se obtienen mediante simulación lógica. En muchos sistemas ATE, y en circuitos en los que se utiliza la autoverificación integrada, los datos de respuesta obtenidos de toda una secuencia de prueba se comprimen en un único valor llamado *identificación*, que se compara con la identificación de un circuito correcto. Esto elimina el gasto de almacenamiento de los vectores de respuesta individuales.

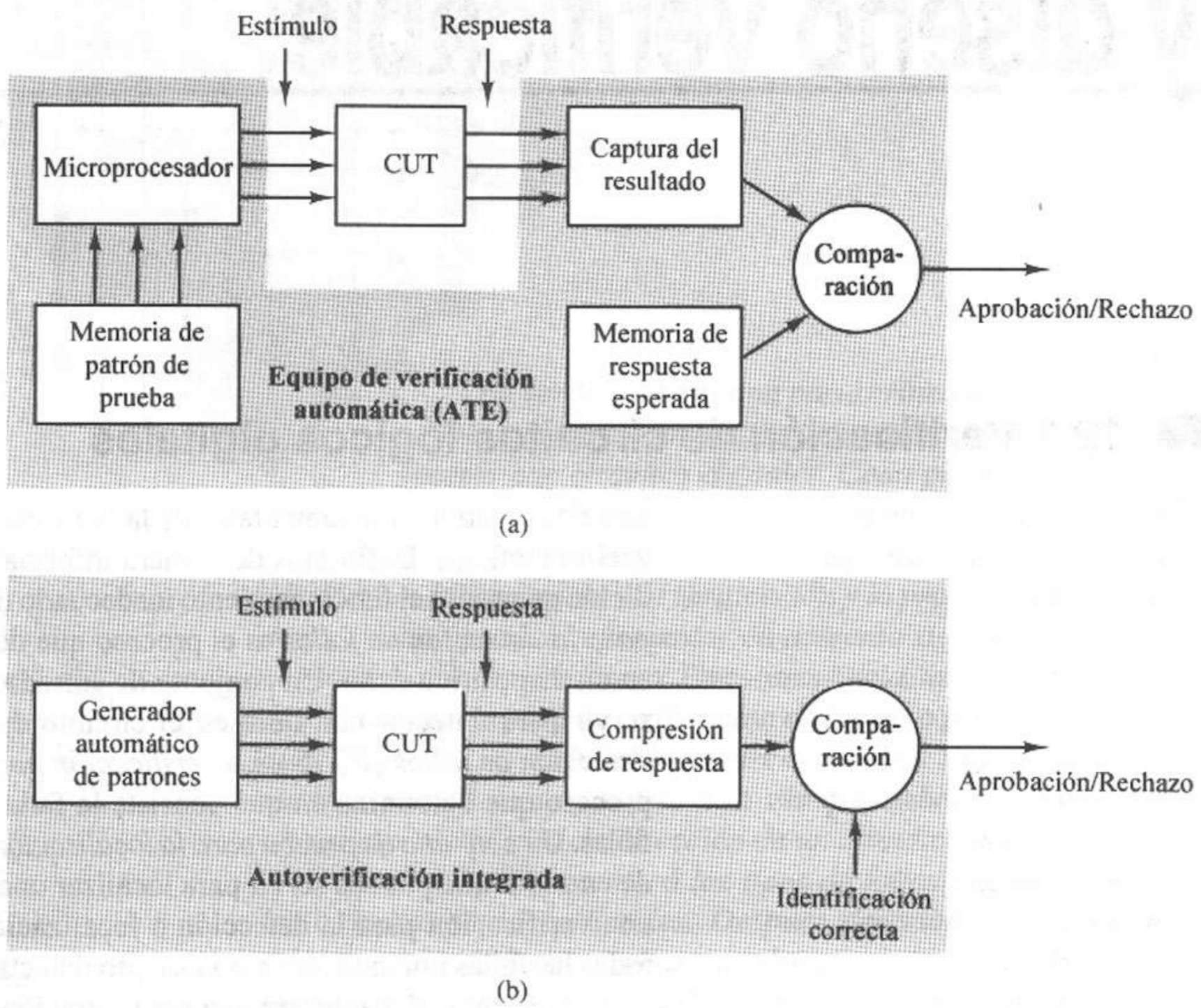


Figura 12.1 Verificación de un circuito lógico digital. **(a)** Configuración de una prueba ATE. **(b)** Configuración de prueba integrada.

12.2 Modelos de fallas

Ahora estudiaremos con detalle el concepto de falla. Las redes lógicas pueden tener fallas causadas por interconexiones rotas o interrumpidas, elementos lógicos incorrectos, voltaje inadecuado, baja inmunidad al ruido, etcétera. Podemos clasificar las fallas de diversas formas. Una condición de falla que no varía con el tiempo es una falla *permanente* o *sólida*. Por otro lado, una falla que aparece y desaparece con el tiempo es una falla *intermitente*. Otras categorías de fallas especifican el efecto de una falla sobre el dispositivo. En este contexto, las *fallas lógicas* hacen que un dispositivo lógico dado funcione como uno completamente distinto. Las fallas *no lógicas* incluyen todas las fallas distintas de las fallas lógicas. Aquí sólo nos ocuparemos de las fallas lógicas sólidas.

Para estudiar el efecto de las fallas sobre un circuito lógico, debemos establecer un modelo de falla. Un modelo popular y útil para la representación de las fallas de los circuitos lógicos es el *modelo de fallas de enclavado* (*stuck–at*). En este modelo, una falla en el circuito se representa mediante un cable, que mantiene el valor 0 lógico (s–a–0) o 1 lógico (s–a–1). La figura 12.2 muestra un circuito con una falla s–a–0. Podemos identificar la falla como el cable ③ s–a–0. Una notación abreviada para la falla es 3/0. Un circuito con falla se puede

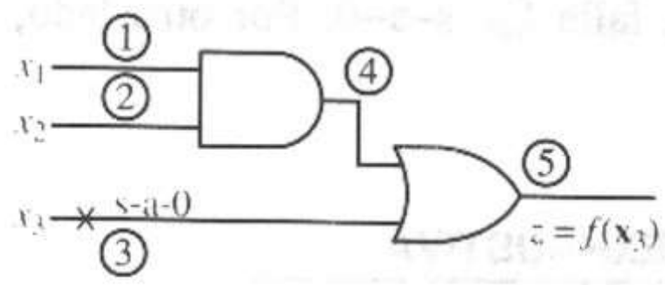


Figura 12.2 Modelo de falla de enclavado para 3 s-a-0.

considerar como un circuito lógico que realiza una función distinta de la realizada por el circuito sin fallas. Adoptaremos una notación funcional estándar para representar los circuitos con fallas. Si $f(\mathbf{x}_n) = f(x_1, x_2, \ldots, x_n)$ representa un circuito sin fallas, $f^{p/d}(\mathbf{x}_n)$ representa el mismo circuito con la falla p/d, donde p es una etiqueta de cable, d puede ser 0 o 1, según si se trata de una falla s–a–0 o s–a–1, y n es el número de variables de entrada. El circuito con fallas de la figura 12.2 tiene la siguiente representación:

$$f^{3/0}(\mathbf{x}_3) = x_1 x_2$$

Una falla 2/1 produce la función:

$$f^{2/1}(\mathbf{x}_3) = x_1 + x_3$$

Una *falla simple* ocurre cuando un único cable está enclavado. Cuando más de un cable está enclavado, tenemos una *falla múltiple*. Los circuitos con r cables tienen $2r$ posibles fallas simples en el modelo de enclavado. Cuando consideramos las fallas múltiples, podemos enumerar $3^r - 1$ fallas. Este número incluye las fallas sencillas como un caso particular de las fallas múltiples.

El modelo de enclavado se justifica por su sencillez y su representación precisa de una clase extensa de fallas que ocurren en los circuitos prácticos. Podemos describir los componentes abiertos o interrumpidos (como transistores o diodos) como los causantes de las fallas de enclavado de un circuito. Los cables rotos y conectados a tierra o a un alto voltaje se pueden representar como fallas de enclavado.

Otras fallas potenciales de los circuitos digitales no son tan fáciles de modelar o detectar. Entre éstas se encuentran las *fallas de puenteo*, que son cortos entre los cables que permiten a una señal en un cable afectar la señal en otro. La lógica programable sufre de una clase única de fallas, llamadas *fallas de intersección*: la ausencia o presencia errónea de diodos o fusibles en las intersecciones de un arreglo lógico. Las fallas de puenteo y de intersección alteran la función lógica realizada por el circuito de manera diferente a las fallas de enclavado.

Muchas fallas son *no lógicas*, pues no alteran la función lógica realizada por el circuito, sino que afectan parámetros del circuito, como los retardos por propagación y los niveles de voltaje/corriente. Las fallas no lógicas requieren pruebas paramétricas especiales y otros métodos de verificación que van más allá del objetivo de este libro.

En este capítulo nos centraremos en las fallas simples de enclavado. Sin embargo, las técnicas que presentaremos son fundamentales para la verificación de otros tipos de fallas.

12.3 Verificación de un circuito lógico combinatorio

Ahora consideraremos el problema de diagnosticar una falla en las redes lógicas combinatorias. El siguiente análisis nos servirá para presentar temas que estudiaremos más adelante.

Sea $f(\mathbf{x}_n)$ la salida de una red lógica que estamos verificando en busca de una posible falla. Es claro que podríamos utilizar como FDTS el conjunto de todas las 2^n entradas posibles a las redes. El uso de todas las entradas posibles para la verificación es una *verificación exhaustiva*, y no es práctica para las redes con un gran número de cables de entrada. Sin embargo, el método es directo, como vemos en el ejemplo siguiente.

Consideremos la red de la figura 12.2. Verificamos dos copias de la red aplicando las ocho combinaciones posibles de entrada a cada copia y observando las respuestas de la red resultante. La tabla 12.1 muestra los resultados de la verificación. Vemos que la copia 1 tiene una falla, pues hemos obtenido algunas respuestas incorrectas. La copia 1 tiene una falla ③ s–a–0. Por otro lado, consideramos que la copia 2 no tiene fallas.

TABLA 12.1 VERIFICACIÓN EXHAUSTIVA

Pruebas			Respuestas	
x_1	x_2	x_3	Copia 1	Copia 2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

La mayor parte de las redes se pueden verificar sin utilizar el método exhaustivo, como veremos en las páginas siguientes al ananlizar la forma de determinar FDTS eficientes para las redes lógicas combinatorias. Un FDTS eficiente es un conjunto de combinaciones de entradas que verifica cualquier falla posible en un conjunto dado de fallas. El conjunto de prueba contiene un número mínimo o casi mínimo de combinaciones de entrada.

12.3.1 Generación de pruebas

Podemos describir la generación de pruebas como el proceso de determinar una prueba para una falla dada en una red dada. Cuando existe más de una de tales pruebas, por lo general se determinan todas las pruebas que se pueden utilizar para detectar la falla. Esta sección contiene un análisis de la generación de pruebas para las fallas de enclavado en las redes lógicas combinatorias. Describiremos dos métodos para la generación de pruebas: el método OR–exclusivo y el método de sensibilización de trayectorias. También incluiremos un análisis de las fallas no verificables y la generación de pruebas para redes con varias salidas.

Sea $f(\mathbf{x}_n)$ una red lógica sin fallas. Una *prueba* T_i para la falla *p/d* es cualquier entrada $\mathbf{x}^j_n$ a la red que satisface la siguiente relación:

$$f(\mathbf{x}^j_n) = \bar{f}^{p/d}(\mathbf{x}^j_n) \tag{12.1}$$

donde el superíndice *j* es el valor decimal de las *n* entradas binarias. Por ejemplo, de la figura 12.2. $\mathbf{x}^1_3 = (001)$ es una prueba para 3/0, pues $f(\mathbf{x}^1_3) = 1 = \overline{0} = \bar{f}^{3/0}(\mathbf{x}^1_3)$. Esta definición se justifica observando que una prueba para una falla dada debe producir una respuesta diferente cuando la falla está presente que cuando

la falla está ausente. Una falla dada puede tener más de un vector de prueba, y un vector de prueba puede verificar más de una falla.

Podemos replantear la condición anterior, que debe ser satisfecha por una entrada para ser una prueba de falla, en términos de la operación OR exclusivo como sigue:

Esta descripción alternativa es la base del primer procedimiento para la generación de pruebas que presentaremos más adelante.

Una *tabla de fallas* es una tabla que exhibe un conjunto de fallas y un conjunto de entradas de prueba. La tabla 12.2 muestra una tabla de fallas que contiene todas las fallas simples y todas las entradas de la red de la figura 12.2. Un 1 en la fila *i*, columna *j*, (*i,j*), indica que la entrada enumerada en la fila *i* es una prueba para la falla enumerada en la columna *j*. Por otro lado, un 0 en (*i,j*) indica que la entrada *i* no es una prueba para la falla *j*. Por ejemplo, la entrada 010 es una prueba para las fallas 1/1, 3/1, 4/1 y 5/1, pero no para 1/0, 2/0, 2/1, 3/0, 4/0 o 5/0.

TABLA 12.2 TABLA DE FALLAS

Pruebas			Fallas									
x_1	x_2	x_3	1/0	1/1	2/0	2/1	3/0	3/1	4/0	4/1	5/0	5/1
0	0	0	0	0	0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0	0	0	1	0
0	1	0	0	1	0	0	0	1	0	1	0	1
0	1	1	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	0	1	0	1	0	1	0	1
1	0	1	0	0	0	0	1	0	0	0	1	0
1	1	0	1	0	1	0	0	0	1	0	1	0
1	1	1	0	0	0	0	0	0	0	0	1	0

Método OR exclusivo

Ahora describiremos un método directo para generar todas las pruebas posibles para una falla dada en una red. Sea $f(\mathbf{x}_n)$ una red sin fallas, y sea *p/d* una falla para la que se desean deducir pruebas. El método se inicia con la construcción de las tablas de verdad de f y de $f^{p/d}$. A continuación, calculamos y registramos $f \oplus f^{p/d}$ para cada fila de las tablas de verdad. La tabla 12.3 muestra estos pasos para las fallas 1/0, 2/1 y 3/0 en la red de la figura 12.2.

Por la definición de prueba para una falla (Ec. 1.2.2), tenemos que las pruebas para la falla *p/d* quedan indicadas por los unos en la columna correspondiente a $f \oplus f^{p/d}$. Por ejemplo, podemos verificar la falla 1/0 sólo mediante la entrada 110, mientras que la falla 3/0 se puede verificar con 001, 011 o 101.

Las pruebas para una falla *p/d* son mintérminos de la función de conmutación $f \oplus f^{p/d}$. Por tanto, al expresar f y $f^{p/d}$ en forma algebraica podemos determinar una expresión que proporcione todas las pruebas para *p/d* aplicando el

TABLA 12.3 MÉTODO DE OR EXCLUSIVO

Pruebas				Funciones realizadas por circuitos con fallas					
x_1	x_2	x_3	f	$f^{1/0}$	$f^{2/1}$	$f^{3/0}$	$f \oplus f^{1/0}$	$f \oplus f^{2/1}$	$f \oplus f^{3/0}$
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	1
0	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	1
1	0	0	0	0	1	0	0	1	0
1	0	1	1	1	1	0	0	0	1
1	1	0	1	0	1	1	1	0	0
1	1	1	1	1	1	1	0	0	0

álgebra booleana sin utilizar las tablas de verdad. Si $F^{p/d}$ representa todas las pruebas para la falla p/d, entonces $F^{p/d} = f \oplus f^{p/d}$. Para la falla 1/0 en la figura 12.2,

$$F^{1/0} = (x_1x_2 + x_3) \oplus (x_3) = x_1x_2\bar{x}_3$$

El mintérmino $x_1x_2\bar{x}_3$ implica la prueba 110. Para 3/0,

$$\begin{aligned} F^{3/0} &= (x_1x_2 + x_3) \oplus (x_1x_2) \\ &= (\bar{x}_1 + \bar{x}_2)x_3 \\ &= \bar{x}_1x_3 + \bar{x}_2x_3 \end{aligned}$$

que tiene los mintérminos $\bar{x}_1\bar{x}_2x_3$, $x_1\bar{x}_2x_3$ y $\bar{x}_1x_2x_3$, los cuales implican las pruebas 001, 101 y 011.

El método algebraico es de particular utilidad cuando se manejan funciones con una gran cantidad de variables. Por ejemplo, consideremos la red de la figura 12.3. (Podemos utilizar una red de este tipo cuando sólo se dispone de compuertas con dos entradas.)

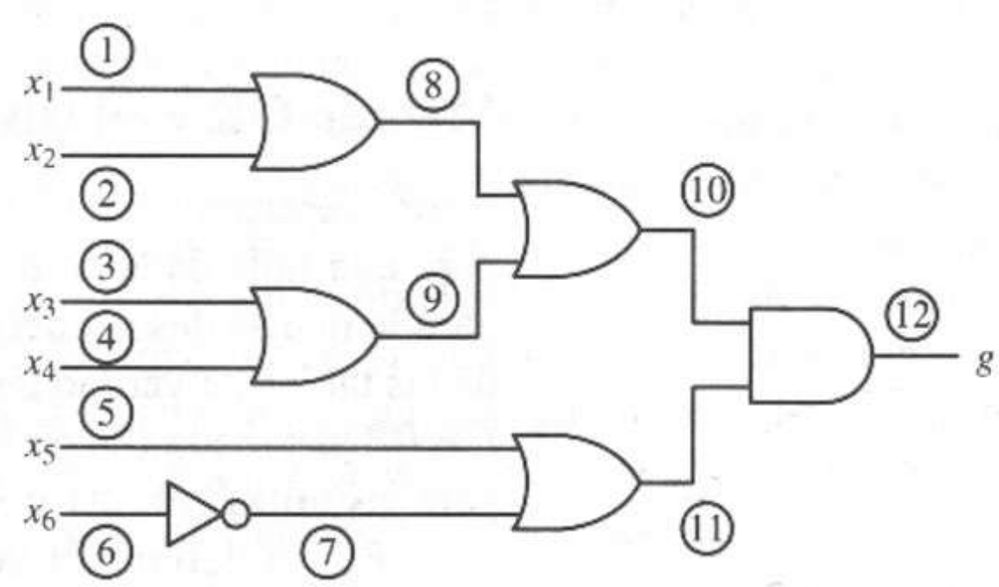


Figura 12.3 Red con *fan-out*.

La función de seis variables correspondiente a esta red es

$$g(\mathbf{x}_6) = (x_1 + x_2 + x_3 + x_4)(x_5 + \bar{x}_6)$$

Consideremos la generación de pruebas para 6/0. El método de la tabla de verdad requiere la formación de tres tablas, con 64 entradas cada una. Es claro que esto requiere mayor esfuerzo que el método algebraico ilustrado a continuación para el mismo caso:

$$\begin{aligned} G^{6/0} &= g(\mathbf{x}_6) \oplus g^{6/0}(\mathbf{x}_6) \\ &= [(x_1 + x_2 + x_3 + x_4)(x_5 + \bar{x}_6)] \oplus [x_1 + x_2 + x_3 + x_4] \\ &= (x_1 + x_2 + x_3 + x_4)\bar{x}_5 x_6 \\ &= x_1\bar{x}_5x_6 + x_2\bar{x}_5x_6 + x_3\bar{x}_5x_6 + x_4\bar{x}_5x_6 \end{aligned}$$

Aquí vemos que existen muchas pruebas para 6/0.

También podemos determinar las pruebas para fallas internas, como 10/1, mediante el método anterior, como sigue:

$$\begin{aligned} G^{10/1} &= g(\mathbf{x}_6) \oplus g^{10/1}(\mathbf{x}_6) \\ &= [(x_1 + x_2 + x_3 + x_4)(x_5 + \bar{x}_6)] \oplus [x_5 + \bar{x}_6] \\ &= \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4(x_5 + \bar{x}_6) \\ &= \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 + \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_6 \end{aligned}$$

Una prueba implicada por $G^{10/1}$ es $x_1 = x_2 = x_3 = x_4 = x_6 = 0$, $x_5 = 1$.

Aunque el método del OR exclusivo es directo, con frecuencia el cálculo es largo con el método de la tabla de verdad y tedioso con el método algebraico. Esto es cierto sobre todo cuando hay que deducir pruebas para las $2r$ fallas sencillas de una red. En la siguiente sección presentaremos un método para la generación de pruebas que supera todas estas dificultades de cómputo.

Método de sensibilización de trayectorias

El método de sensibilización de trayectorias selecciona primero una trayectoria desde el sitio de la falla hasta la salida de la red. Entonces, las entradas se eligen de modo que los valores lógicos de las líneas de la trayectoria sean una función de la falla. Una trayectoria ha sido *sensibilizada* del sitio de la falla hasta la salida cuando se cumple esta condición. Por ejemplo, consideremos la compuerta OR de la figura 12.4a. La salida z depende del valor lógico en la línea ① cuando el valor lógico en la línea ② es 0. Sea 1/0 la falla para la cual queremos

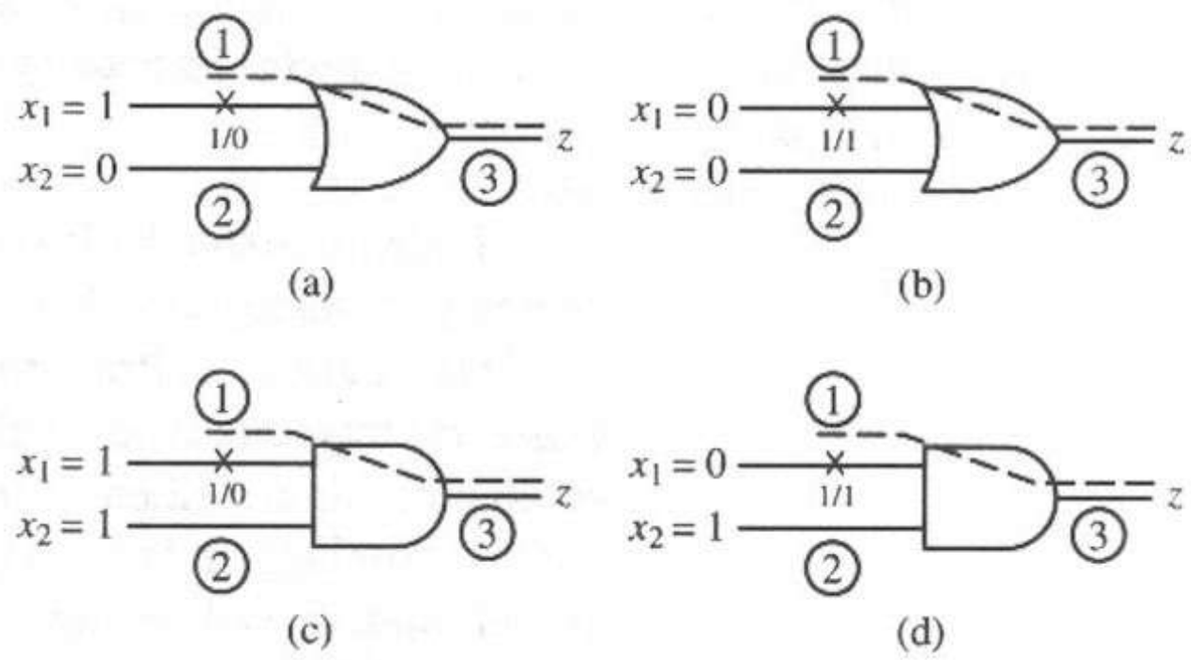


Figura 12.4 Sensibilización de trayectorias para compuertas. **(a)** Compuerta OR con 1/0. **(b)** Compuerta OR con 1/1. **(c)** Compuerta AND con 1/0. **(d)** Compuerta AND con 1/1.

deducir una prueba. La entrada $x_1 = 1, x_2 = 0$ hace a la salida $z = 1$ cuando 1/0 no está presente y $z = 0$ cuando la falla está presente. Por tanto, $x_1 = 1$, $x_2 = 0$ sensibiliza una trayectoria desde la falla potencial 1/0 hasta la salida y es una prueba para 1/0. La entrada $x_1 = x_2 = 0$ sensibiliza una trayectoria para la falla 1/1, como muestra la figura 12.4b, y es una prueba para 1/1. Podemos sensibilizar las trayectorias en una compuerta AND, como muestran las figuras 12.4c y d. En general, las trayectorias se sensibilizan a través de las compuertas lógicas básicas como en la figura 12.5.

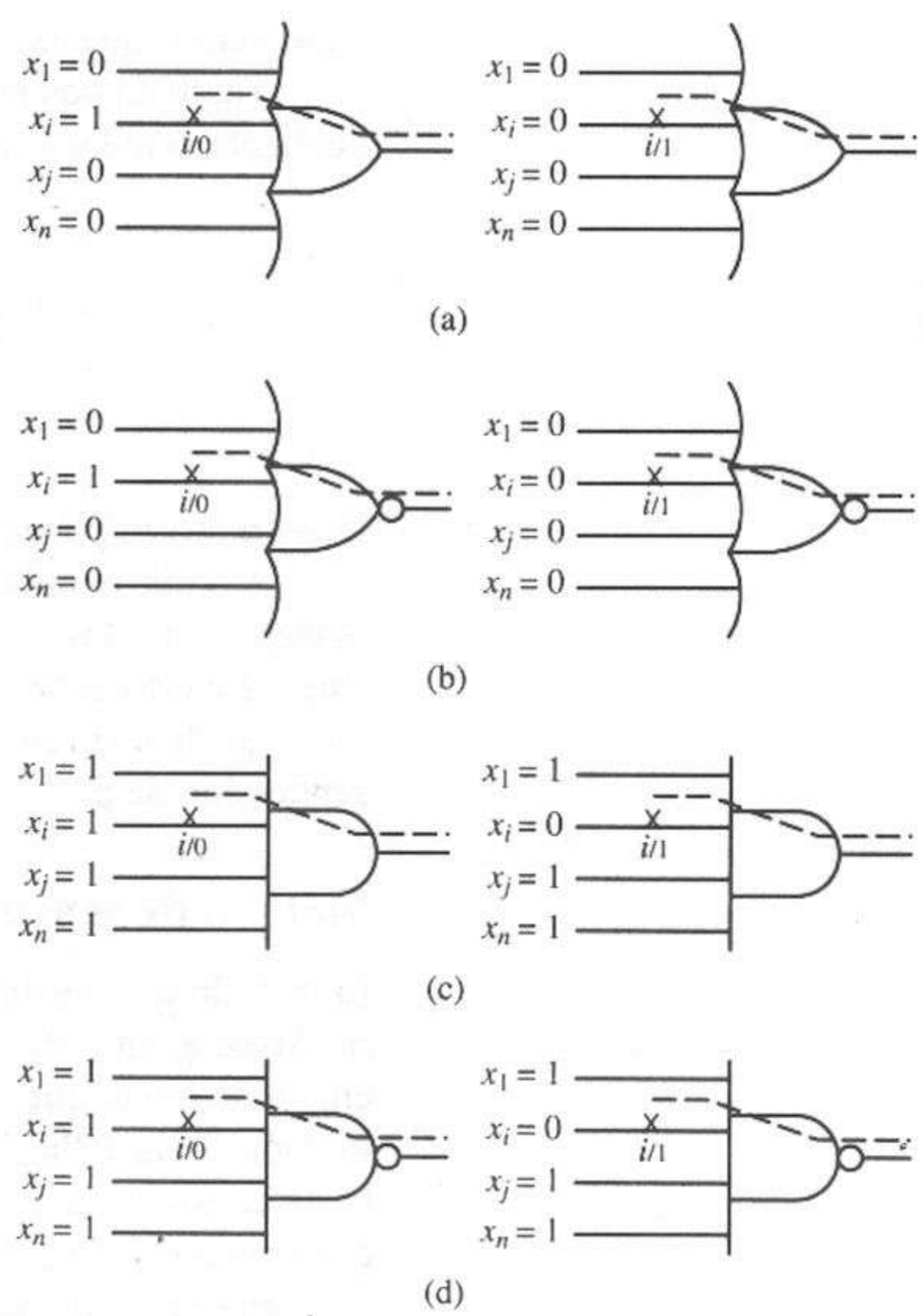
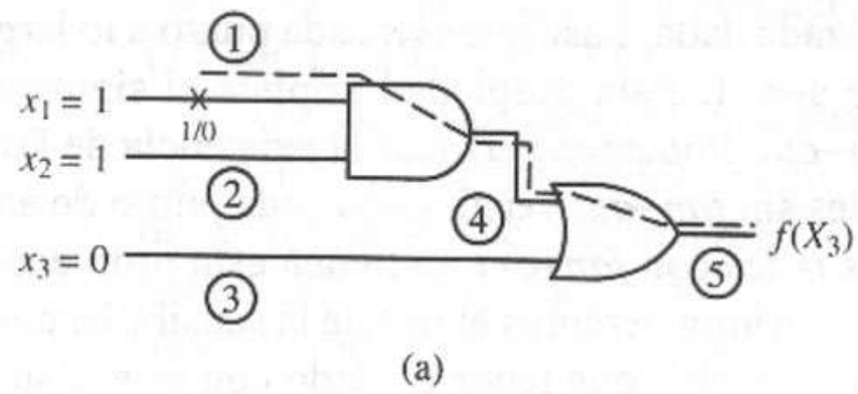


Figura 12.5 Sensibilización de trayectorias en compuertas lógicas comunes. **(a)** Compuerta OR. **(b)** Compuerta NOR. **(c)** Compuerta AND. **(d)** Compuerta NAND.

Podemos extender la sensibilización de trayectorias para utilizarla en redes de compuertas lógicas. Ahora describiremos esta extensión para la red sin *fan-out* de la figura 12.6. Primero, determinaremos una prueba para la falla 1/0. Sólo existe una trayectoria desde el punto ① hasta la salida de la red y la sensibilizamos como en la figura 12.6a. Necesitamos la entrada $x_1 = 1$ para establecer el valor lógico apropiado en la línea ① y verificar una falla de enclavado en 0, mientras que el valor $x_2 = 1$ es necesario para sensibilizar la trayectoria a través de la compuerta AND. Para extender la trayectoria de la compuerta OR a la salida es necesaria la entrada $x_3 = 0$. Por tanto, hemos establecido la prueba 110 para la falla 1/0.



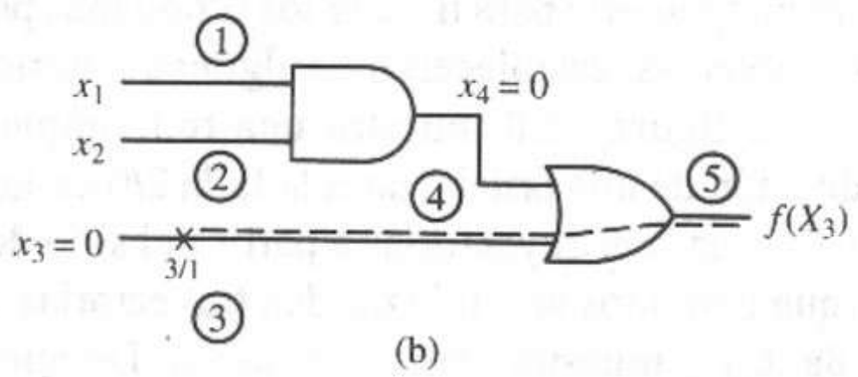


Figura 12.6 Sensibilización de trayectorias para redes. **(a)** Trayectoria para 1/0. **(b)** Trayectoria para 3/1.

Determinemos ahora una prueba para la falla 3/1. Necesitamos la entrada $x_3 = 0$ para inicializar la trayectoria, ya que estamos considerando una falla s–a–1. Sensibilizamos una trayectoria hasta la salida estableciendo el valor lógico 0 en el punto ④. Este cable aparece como $x4$ en la figura 12.6b. Con $x_3 = x_4 = 0$, sensibilizamos la trayectoria necesaria; esto se conoce como el paso de *rastreo hacia adelante* del método de sensibilización de trayectorias. Para hacer $x_4 = 0$, x_1 y x_2 deben ser 00, 01 o 10. El hecho de fijar x_1 y x_2 es el paso de *rastreo hacia atrás* del método. Así, hemos determinado las tres pruebas 000, 010 o 100 para 3/1.

Podemos resumir el método de sensibilización de trayectorias como sigue:

1. Seleccionamos la falla para la cual queremos determinar las pruebas y elegimos una trayectoria desde el sitio de la falla hasta la salida de la red.
2. Sensibilizamos la trayectoria (rastreo hacia adelante).
3. Establecemos las entradas de la red según lo requerido por el paso 2 (rastreo hacia atrás).

Otra aplicación útil de la sensibilización de trayectorias, además de la generación de pruebas, es la determinación del conjunto de fallas verificadas por una entrada dada. Consideremos de nuevo la red de la figura 12.6, con la entrada 110. Para mayor claridad, hemos trazado la red de nuevo en la figura 12.7. Hemos rotulado las líneas a lo largo de la trayectoria desde la entrada $x1$ hasta la salida de la red como $1 \rightarrow 0$, lo que indica: (valor lógico *sin* fallas) $\rightarrow$ (valor lógico con falla). Por tanto, la entrada 110 verifica las fallas 1/0, 4/0 y 5/0. La trayectoria de ② a ④ también es sensibilizada por $x_1 = 1$, así que la entrada 110 también verifica 2/0.

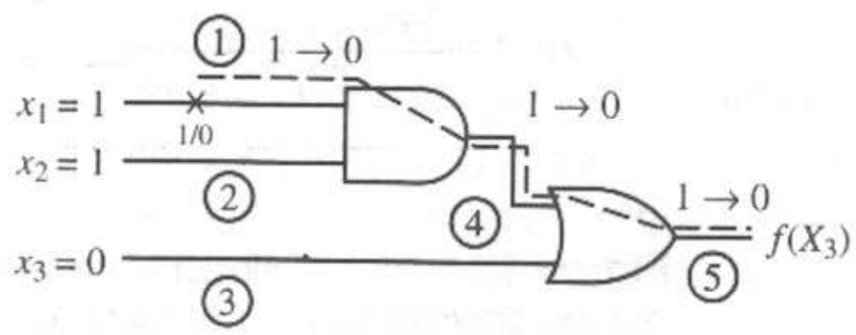


Figura 12.7 Propagación a lo largo de una trayectoria sensibilizada.

Podemos resumir el análisis anterior como sigue. Para una trayectoria sensibilizada dada, buscamos en cada punto a lo largo de la trayectoria una falla s–a–0 o s–a–1. Esta propiedad implica el siguiente hecho acerca de las redes sin *fan–out*. Podemos verificar la existencia de fallas simples s–a–0 y s–a–1 en las redes sin *fan–out* verificando cada punto de entrada a las fallas s–a–0 y s–a–1. Las redes con *fan–out* no tienen esta propiedad.

Ahora veremos el uso de la sensibilización de trayectorias en las redes con *fan–out*. Hay que tener cuidado con este caso. Presentaremos tres ejemplos de redes con *fan–out* para ilustrar los problemas potenciales. Después de identificar los problemas, estableceremos algunos criterios para evitar estos problemas.

La figura 12.8 muestra una red simple con *fan–out*. Consideremos la deducción de una prueba para la falla 2/0 en la red. Como la línea ② tiene *fan–out*, existen dos trayectorias a partir del sitio de la falla hasta la salida de la red, así que podemos sensibilizar dos trayectorias simples y una doble. Las figuras 12.8a, b y c muestran estos tres casos. De nuevo utilizamos la notación $1 \rightarrow 0$

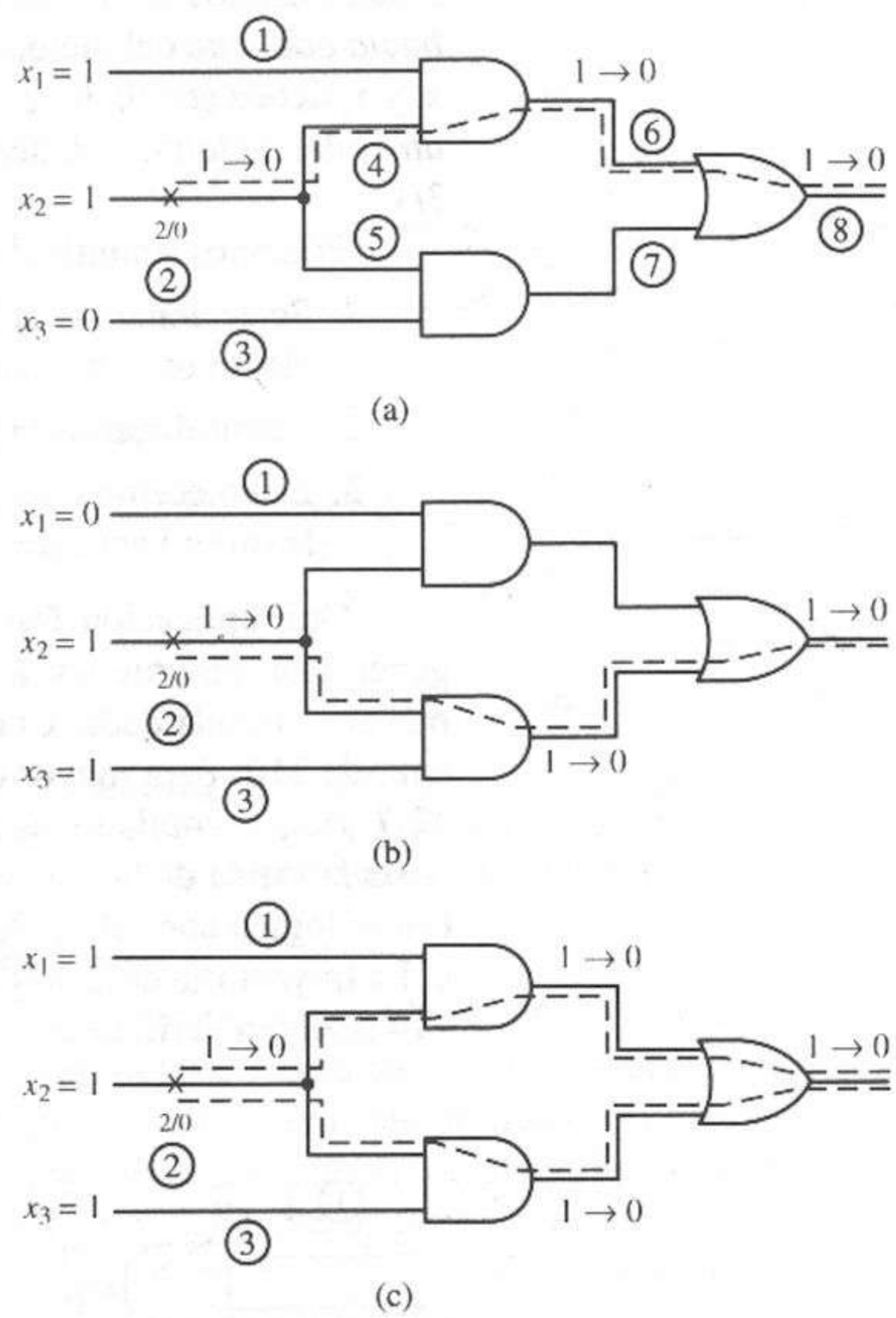


Figura 12.8 Sensibilización de trayectorias en una red con *fan-out*. **(a)** Trayectoria sencilla dada por 110. **(b)** Trayectoria sencilla dada por 011. **(b)** Trayectorias dobles dadas por 111.

para mostrar que cada caso produce una prueba para 2/0. Por tanto, 110, 011 y 111 son pruebas para 2/0.

Deduciremos ahora las pruebas para la falla 2/0 en la red con *fan–out* de la figura 12.9. Observe que éste es un multiplexor simple 2 a 1, con x_2 como el bit de control de selección. De nuevo, existen dos trayectorias sencillas y una doble para la falla. Sin embargo, observe que al sensibilizar la trayectoria doble la salida de la red es 1, sin importar que la falla 2/0 esté presente o no. Por tanto, la entrada 111 no es una prueba para la falla, pero las entradas 110 y 011 sí son pruebas.

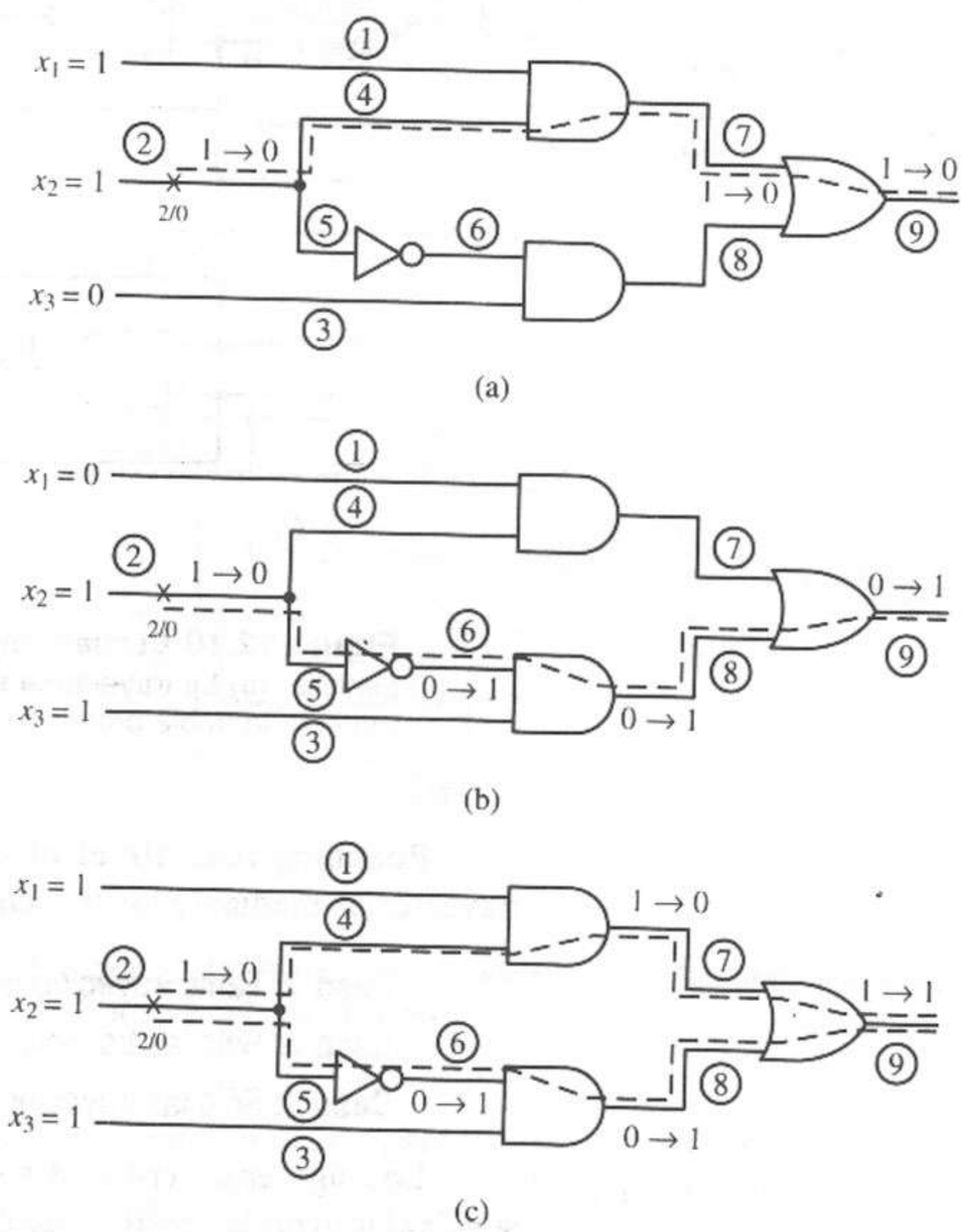
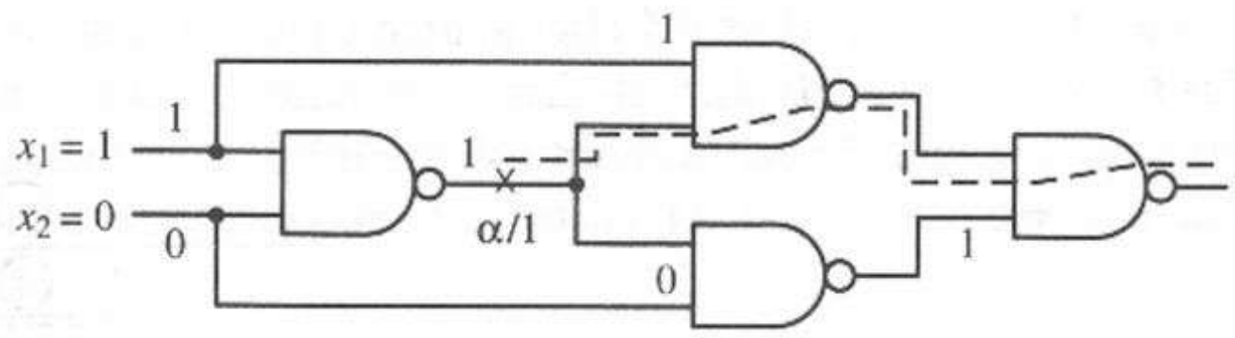


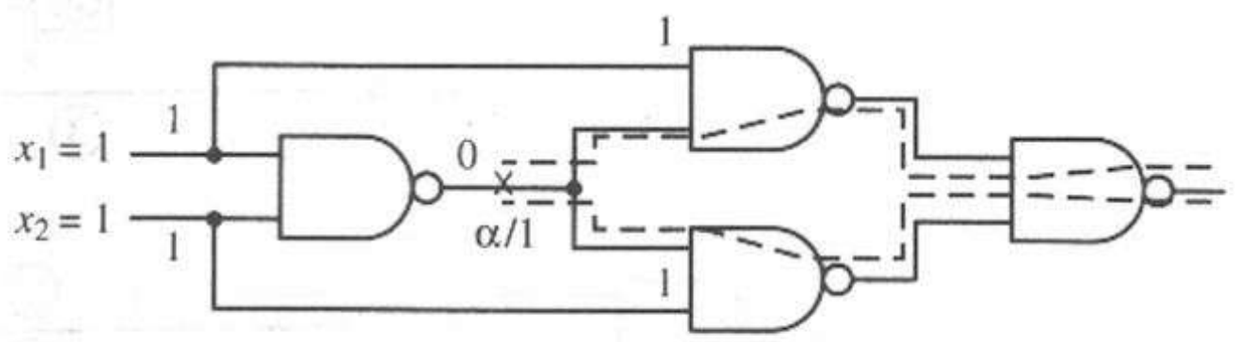
Figura 12.9 Red de sensibilización de trayectorias con paridad desigual. **(a)** Trayectoria sencilla dada por 110. **(b)** Trayectoria sencilla dada por 011. **(c)** La trayectoria doble dada por 111 no es una prueba.

Los dos últimos ejemplos muestran que en un caso una trayectoria múltiple produce una prueba, pero en el otro no. Las trayectorias sencillas produjeron pruebas en ambos ejemplos. Una generalización que podríamos considerar después de ver estos ejemplos es que sólo debemos sensibilizar trayectorias sencillas al deducir pruebas. Sin embargo, la red de la figura 12.10 muestra un contraejemplo de tal generalización. Esta red se utiliza con frecuencia en vez de una compuerta OR exclusiva, ya que sólo utiliza compuertas NAND sencillas

de dos entradas sin inversores adicionales. La figura 12.10a muestra un intento por obtener una prueba para $\alpha/1$ mediante la sensibilización de una trayectoria simple. Aparecen condiciones contradictorias para las entradas de la red, ya que no es posible aplicar 0 a α y mantener un 0 en la entrada inferior de la compuerta NAND que aparece fuera de la trayectoria en la figura. Obtenemos resultados similares al sensibilizar otra trayectoria simple. La figura 12.10b muestra la deducción exitosa de una prueba al sensibilizar la trayectoria doble.



(a)



(b)

Figura 12.10 Contraejemplo de una sensibilización de trayectorias sencilla. **(a)** La trayectoria sencilla no produce una prueba. **(b)** La trayectoria doble produce una prueba.

Podemos resumir el efecto del *fan–out* sobre la sensibilización de trayectorias mediante los tres casos siguientes:

Caso 1: Tanto trayectorias sencillas como las múltiples producen pruebas.

Caso 2: Sólo las trayectorias sencillas producen pruebas.

Caso 3: Sólo las trayectorias múltiples producen pruebas.

Los siguientes criterios son útiles para evitar los problemas que podrían surgir al utilizar la sensibilización de trayectorias para deducir pruebas relativas a una falla en las redes con *fan–out*.

1. Intente deducir las pruebas utilizando sólo trayectorias sencillas. Continúe con el paso 2 sólo si no pudo determinar pruebas.
2. Intente deducir pruebas utilizando sólo trayectorias múltiples. Verifique la validez de cada prueba potencial. Deténgase al determinar una prueba.
3. Si existen m trayectorias simples posibles, deberá examinar todas las combinaciones posibles de 2, 3,..., m trayectorias antes de concluir que no existe prueba alguna.

Un procedimiento basado en estos criterios no garantiza determinar todas las pruebas para una falla dada.

12.3.2 Fallas no verificables

Una falla *p/d* es *verificable* si y sólo si existe al menos una prueba para la falla. Todas las fallas consideradas en los ejemplos anteriores han sido verificables. Sin embargo, como demostraremos a continuación, no todas las fallas son verificables. Éstas son las fallas *no verificables.*

Considere la falla 8/1 en la red de la figura 12.11a. Podemos mostrar de manera algebraica que $f_\alpha^{8/1} = f_\alpha$. Entonces, por el método de OR exclusivo, $F_\alpha^{8/1} = 0$, lo que implica que no existen pruebas para esta falla; es decir, la falla 8/1 no es verificable.

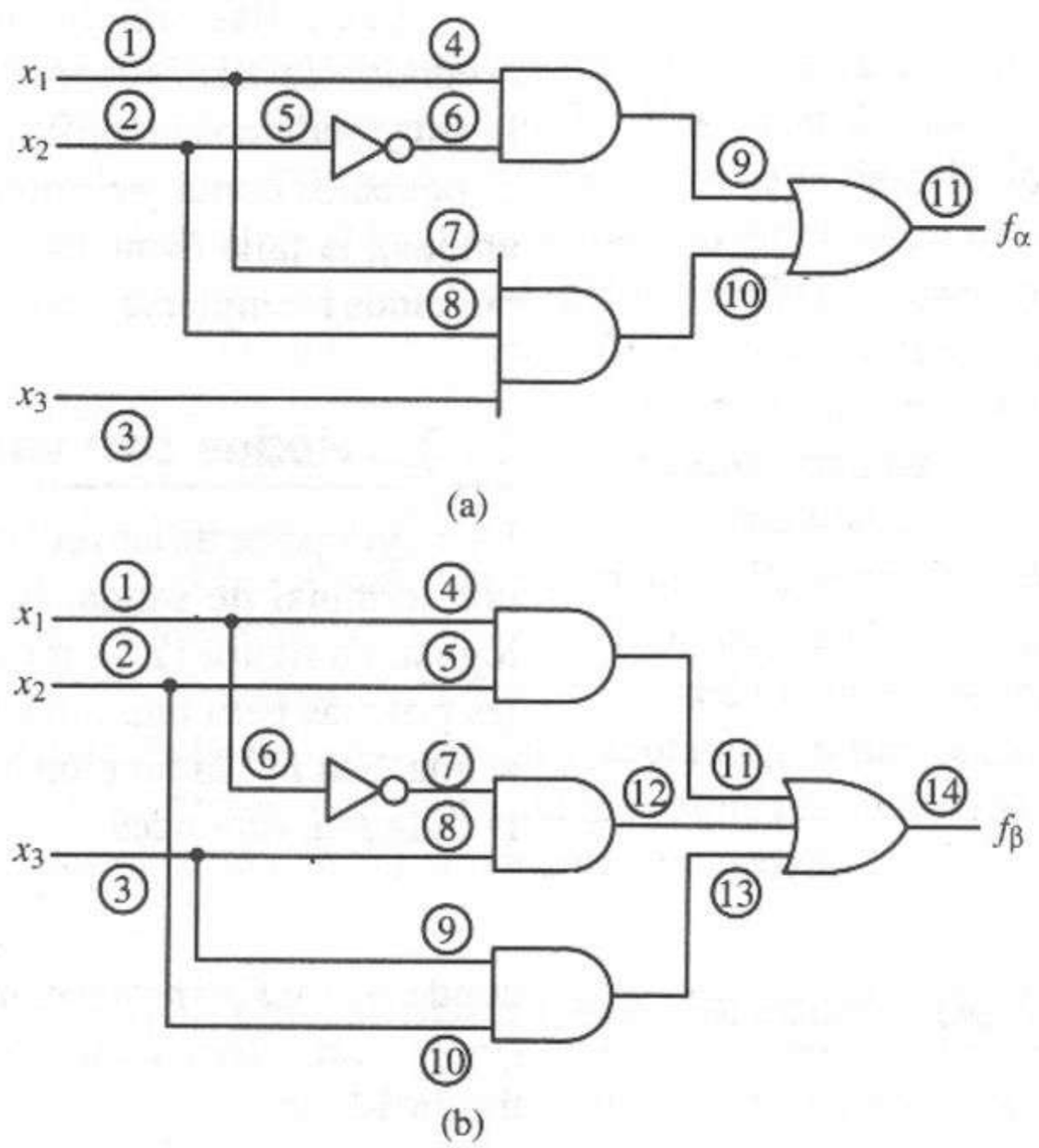


Figura 12.11 Redes con redundancia. **(a)** Redundancia literal. **(b)** Redundancia de términos.

Una situación similar ocurre para la falla 13/0 en la red de la figura 12.11b. La función $f_\beta^{8/1} = f_\beta$ y, por tanto, $F_\beta^{13/0} = 0$, de modo que la falla 13/0 no es verificable. Sin embargo, observe que $F_\alpha^{8/0}$ y $F_\beta^{13/1}$ son distintos de cero, como mostramos a continuación:

$$\begin{aligned} F_\alpha^{8/0} &= (x_1\bar{x}_2 + x_1x_2x_3) \oplus (x_1\bar{x}_2) \\ &= x_1x_2x_3 \\ F_\beta^{13/1} &= (x_1x_2 + \bar{x}_1x_3 + x_2x_3) \oplus (1) \\ &= x_1\bar{x}_2 + \bar{x}_1\bar{x}_3 \end{aligned}$$

Por tanto, 8/0 en f_α y 13/1 en f_β son verificables.

Así, surge la pregunta de cómo identificar las fallas no verificables. Podemos encontrar la respuesta en el estudio de la redundancia en las redes lógicas. Una red tiene *redundancia* si y sólo si existe una línea en la red que puede

cortarse y reemplazarse por una constante lógica adecuada (0 o 1) sin modificar la función realizada por la red. Una red sin redundancia es una *red no redundante*. Las redes de la figura 12.11 tienen redundancia, como mostraremos a continuación.

Podemos cortar y reemplazar la línea ⑧ de la red de la figura 12.11 por una entrada constante 1 de la compuerta AND sin modificar la función lógica f_α. Podemos justificar esto fácilmente usando álgebra booleana. Por tanto, no necesitamos la literal x_2 en la expresión y tenemos una redundancia en la red.

En la figura 12.11b podemos cortar la línea ⑬ y reemplazarla por un 0 lógico sin modificar la función realizada por la red. Por tanto, la línea es redundante.

Las fallas simples no verificables sólo pueden ocurrir en redes con redundancia. Las fallas no verificables corresponden a las líneas redundantes de la red y podemos identificarlas como sigue. Sea i una línea redundante de la red. Si podemos cortar y reemplazar i por un valor lógico constante v (0 o 1, pero no ambos), la falla i/v no es verificable. Sin embargo, la falla $i/\bar{v}$ es verificable. Si podemos reemplazar i por v y por $\bar{v}$, las dos fallas i/v y $i/\bar{v}$ no son verificables.

12.3.3 Redes con varias salidas

La mayor parte de las redes lógicas que se presentan en la práctica tienen más de una terminal de salida. En otras palabras, la red realiza más de una función lógica. La figura 12.12 muestra una red con dos terminales de salida. Al generar las pruebas para una falla en esta red debemos considerar ambas funciones de salida. Sea $F^{p/d}$ la función booleana que describe todas las pruebas posibles para la falla p/d. Entonces

$$F^{p/d} = F_1^{p/d} + F_2^{p/d}$$

donde $F_1^{p/d}$ y $F_2^{p/d}$ representan todas las pruebas para la falla p/d en las terminales f_1 y f_2, respectivamente. Por ejemplo, consideremos las fallas 1/0 y 3/1 de la figura 12.12:

$$\begin{aligned} F^{1/0} &= F_1^{1/0} + F_2^{1/0} \\ &= [(x_1x_2 + x_2x_3) \oplus (x_2x_3)] + [(x_2x_3 + x_3x_4) \oplus (x_2x_3 + x_3x_4)] \\ &= [x_1x_2\bar{x}_3] + [0] \\ &= x_1x_2\bar{x}_3 \end{aligned}$$

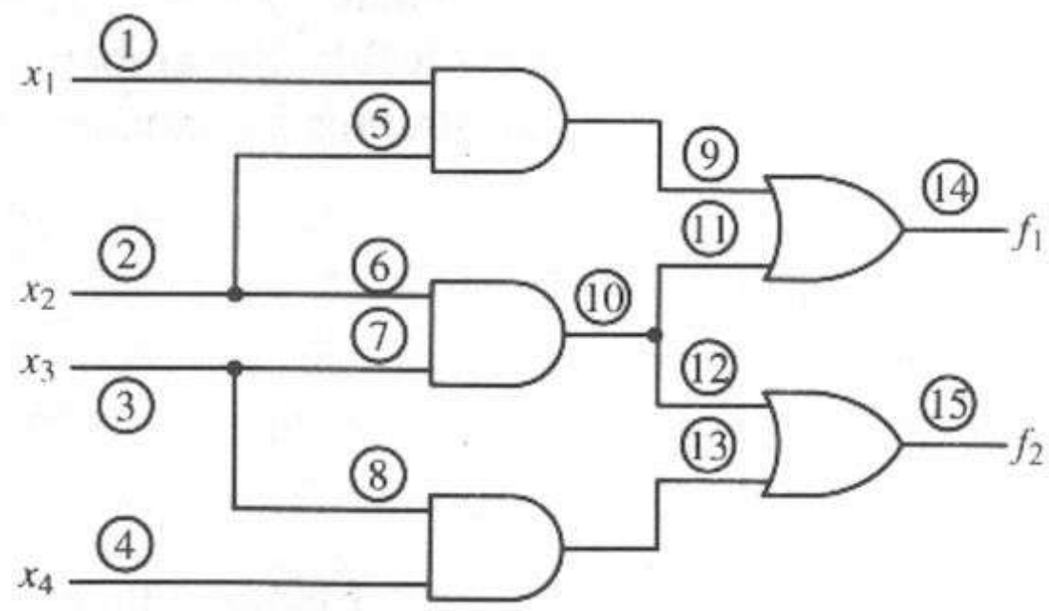


Figura 12.12 Red con varias salidas.

$F_2^{1/0} = 0$, pues la falla 1/0 no puede ser observada en f_2.

$$\begin{aligned} F^{3/1} &= F_1^{3/1} + F_2^{3/1} \\ &= [(x_1x_2 + x_2x_3) \oplus (x_1x_2 + x_2)] + [(x_2x_3 + x_3x_4) \oplus (x_2 + x_4)] \\ &= [\bar{x}_1x_2\bar{x}_3] + [x_2\bar{x}_3 + \bar{x}_3x_4] \\ &= x_2\bar{x}_3 + \bar{x}_3x_4 \end{aligned}$$

En general, para una red con m terminales de salida, las pruebas para la falla p/d están dadas por:

$$F^{p/d} = F_1^{p/d} + F_2^{p/d} + \cdots + F_m^{p/d}$$

Existen muchos algoritmos para generar pruebas para fallas de los circuitos lógicos combinatorios; la mayor parte de ellos están diseñados de modo que produzcan pruebas en forma automática a partir de la descripción del circuito. Por ejemplo, el *algoritmo–D* [20] es un método automático de sensibilización de trayectorias que utiliza valores lógicos especiales para representar las condiciones de error: D representa una señal cuyo valor correcto debería ser 1 pero incorrectamente es 0, mientras que $\bar{D}$ representa la condición de error opuesta. El algoritmo D determina de manera automática las entradas requeridas para producir D (o $\bar{D}$) en el sitio de una falla, y las entradas adicionales necesarias para propagar D o $\bar{D}$ hasta una salida principal. De manera análoga, LASAR (Logic Automated Stimulus and Response) [20] propaga las condiciones de error utilizando valores "forzados" y "críticos" en cada compuerta lógica. PODEM [20] utiliza un método de ramificación y acotamiento para examinar todas las combinaciones de entrada, considerando los valores de una señal de entrada por turno para determinar si pueden conducir a una prueba.

12.3.4 Conjuntos de pruebas para detección de fallas

En la sección anterior presentamos dos métodos de determinación de pruebas para una falla simple verificable s–a–0 o s–a–1. En esta sección analizaremos el problema de seleccionar un FDTS para una red dada. Un FDTS para una red es *completo* si existe al menos una prueba en el conjunto para cada falla posible de la red. Un FDTS *mínimo* es un FDTS completo que contiene el menor número de pruebas para cualquier FDTS completo. La red de la figura 12.8 tiene el siguiente FDTS mínimo:

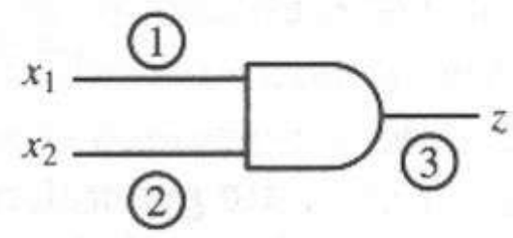
$$\text{FDTS}_m = \{010, 011, 101, 110\}$$

Deduciremos este conjunto de prueba al final de esta sección.

En los párrafos siguientes consideraremos los métodos para elegir FDTS mínimos. Y describiremos la elección de un FDTS casi mínimo. Suponemos que la tabla de fallas de una red se conoce antes de elegir un FDTS. Podemos utilizar cualquiera de los métodos presentados en la sección anterior para obtener la tabla de fallas. Sin embargo, la tabla no tiene que contener todas las fallas posibles de una red al utilizarla en el proceso de selección. Podemos eliminar fallas de la tabla si son equivalentes o dominan a otra falla de la tabla.

Consideremos la compuerta AND de la figura 12.13a. La figura 12.13b muestra su tabla de fallas completa. La tabla nos permite ver que las fallas 1/0, 2/0 y 3/0 tienen exactamente el mismo conjunto de prueba: {11}. Las fallas con el mismo conjunto de prueba son *equivalentes*. Como el conjunto de pruebas

para una de estas fallas detectará las demás, podemos eliminar todas las fallas, excepto una, de cualquier conjunto de fallas equivalentes en una tabla de prueba.



(a)

Pruebas		Fallas					
x_1	x_2	1/0	1/1	2/0	2/1	3/0	3/1
0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	1
1	0	0	0	0	1	0	1
1	1	1	0	1	0	1	0

(b)

Pruebas		Fallas		
x_1	x_2	1/0	1/1	2/1
0	0	0	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

(c)

Figura 12.13 Reducción de la tabla de fallas. **(a)** Compuerta AND. **(b)** Tabla de fallas completa. **(c)** Tabla de fallas reducida.

A continuación, observemos que el conjunto de prueba para la falla 1/1, {01}, es un subconjunto del conjunto de prueba para la falla 3/1, {01,10,00}. Decimos que una falla f_1 *domina* a otra falla f_2 si el conjunto de prueba de f_2 es un subconjunto propio del conjunto de prueba de f_1. En este caso, podemos eliminar f_1 (la falla dominante) de la tabla de fallas, pues cualquier prueba para f_2 también será una prueba para f_1.

En el caso de la compuerta AND de la figura 12.13a, la eliminación de fallas mediante la equivalencia y dominancia de fallas nos conduce a la tabla de fallas reducida de la figura 12.13c. Podemos realizar deducciones similares para los demás tipos de compuertas lógicas. Debido a las reducciones, sólo hay que considerar las fallas en los puntos de verificación de una red. Los *puntos de verificación de una red* son los cables de la red que satisfacen alguna de las siguientes descripciones:

1. Todos los cables de entrada que no son tallos de un *fan-out*

2. Todos los cables de la red que son ramas de un *fan-out*

El término *tallo de un fan–out* se refiere al cable que precede al punto de *fan-out*, y el término *ramas del fan–out* se refiere a los cables que salen del punto de

fan–out. Los puntos de verificación para la red de la figura 12.8 son 1, 3, 4 y 5. La tabla 12.4 muestra la tabla de fallas correspondiente.

TABLA 12.4 TABLA DE FALLAS QUE MUESTRA SÓLO FALLAS EN EL PUNTO DE VERIFICACIÓN PARA LA FIGURA 12.8

Pruebas			Fallas							
x_1	x_2	x_3	1/0	1/1	3/0	3/1	4/0	4/1	5/0	5/1
0	0	0								
0	0	1								1
0	1	0		1		1				
0	1	1			1				1	
1	0	0						1		
1	0	1						1		1
1	1	0	1				1			
1	1	1								

Podemos elegir un conjunto mínimo de prueba para la detección de fallas a partir de una tabla de fallas, seleccionando el mínimo número de entradas que cubre todas las fallas. Este proceso es equivalente al paso de selección de implicantes primos del procedimiento de Quine–McCluskey, en el que elegimos el número mínimo de implicantes primos que cubren todos los mintérminos. En nuestro caso, podemos aplicar en forma directa el procedimiento de selección descrito en el capítulo 3 para el procedimiento de Quine–McCluskey y no lo repetiremos. Al aplicar el procedimiento a la tabla 12.4, obtenemos {010, 011, 101, 110} como conjunto mínimo de prueba.

La selección de un conjunto mínimo de prueba puede ser largo en el caso de las tablas de fallas con un tamaño de moderado a grande. Por tanto, a menudo es más práctico utilizar procedimientos para elegir conjuntos de prueba que no necesariamente sean mínimos. Como veremos más adelante, tales procedimientos proporcionan muchas veces un conjunto mínimo. El procedimiento cercano al mínimo aquí descrito se basa en el cálculo de un peso para cada entrada de la tabla de fallas. El *peso* de una entrada determinada es el número de fallas verificadas por la entrada. Elegimos las entradas para el conjunto de prueba cercano al mínimo mediante el procedimiento siguiente:

Paso 1. Calcular el peso de cada entrada (fila) de la tabla de fallas.

Paso 2. Elegir la entrada de peso máximo. Se elige en forma arbitraria si más de una entrada tiene el peso máximo.

Paso 3. Reducir la tabla de fallas eliminando la entrada elegida y todas las fallas cubiertas por esa entrada.

Paso 4. Calcular de nuevo el peso de cada entrada de la tabla de fallas reducida.

Paso 5. El procedimiento termina cuando todas las entradas de la tabla reducida tienen peso nulo. En caso contrario, se repiten los pasos 2 a 5.

EJEMPLO 12.1

Ilustrar el procedimiento mediante otro ejemplo de circuito cuya tabla de fallas aparece en la tabla 12.5.

TABLA 12.5 TABLA DE FALLAS CON PESOS

	Fallas									
Pruebas	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Pesos**
T_1				1			1			2
T_2							1	1		2
T_3				1	1					2
T_4	1				1	1				3
T_5	1		1			1			1	4
T_6		1	1					1	1	4

Paso 1. Las entradas reciben los siguientes pesos: T_1, 2; T_2, 2; T_3, 2; T_4, 3; T_5, 4; T_6, 4.

Paso 2. Elegimos T_5.

Paso 3. La tabla 12.6a muestra la tabla de fallas reducida.

TABLA 12.6 PROCESO DE REDUCCIÓN DE UNA TABLA DE FALLAS (a) T_5 ELIMINADO (b) T_1 ELIMINADO (c) T_6 ELIMINADO (d) T_3 ELIMINADO

	Fallas					
Pruebas	**2**	**4**	**5**	**7**	**8**	**Pesos**
T_1		1		1		2
T_2				1	1	2
T_3		1	1			2
T_4			1			1
T_6	1				1	2

(a)

	Fallas			
Pruebas	**2**	**5**	**8**	**Pesos**
T_2			1	1
T_3		1		1
T_4		1		1
T_6	1		1	2

(b)

	Fallas	
Pruebas	**5**	**Pesos**
T_2		0
T_3	1	1
T_4	1	1

(c)

	Fallas
Pruebas	**Pesos**
T_2	0
T_4	0

(d)

Paso 4. La tabla reducida 12.6a muestra los nuevos pesos.

Paso 5. Repetimos los pasos 2 a 5.

Paso 2. Elegimos T_1.

Paso 3. La tabla de fallas reducida aparece en la tabla 12.6b.

Paso 4. Los nuevos pesos aparecen en la tabla reducida 12.6b.

Paso 5. Repetimos los pasos 2 a 5.

Paso 2. Elegimos T_6.

Pasos 3 y 4. Ver tabla 12.6c.

Paso 5. Repetimos los pasos 2 a 5.

Paso 2. Elegimos T_3.

Pasos 3 y 4. Ver tabla 12.6d.

Paso 5. Terminamos. El conjunto de prueba es $\{T_1, T_3, T_5, T_6\}$.

La opción de determinar un conjunto mínimo de prueba o determinar un conjunto de prueba cercano al mínimo varía en cada caso. Cuando se necesita el conjunto de prueba más eficiente para minimizar el tiempo de verificación, hay que determinar un conjunto mínimo de prueba, pero, cuando el tiempo de selección de la prueba es más importante que el tiempo de verificación el objetivo deberá ser un conjunto de prueba cercano al mínimo.

12.3.5 Localización y diagnóstico de fallas

Podemos describir el problema de localizar una falla dada en una red como la capacidad para distinguir la falla dada de las demás fallas posibles que pueden ocurrir en la red. Dos fallas son *distinguibles* si y sólo si existe al menos una prueba de detección de la falla para una de las fallas que no sea una prueba para la otra falla. La figura 12.6 muestra que las fallas 1/1 y 2/1 son distinguibles. Entre las demás fallas distinguibles de la figura 12.6 están 3/0 y 5/0. Dos o más fallas son *indistinguibles* o *equivalentes* si y sólo si tienen exactamente el mismo conjunto de pruebas para detección de fallas. En la figura 12.6, {1/0, 2/0, 4/0} y {3/1, 4/1, 5/1} son dos conjuntos de fallas indistinguibles.

Sea $F^{i/d_i-j/d_j}$ una función booleana que representa todas las pruebas que distinguen las fallas i/d_i y j/d_j en una red que realiza $f(\mathbf{x}_n)$. Entonces

$$F^{i/d_i-j/d_j} = F^{i/d_i} \oplus F^{j/d_j} \tag{12.3}$$

donde F^{i/d_i} y F^{j/d_j} describen todas las pruebas para i/d_i y j/d_j, respectivamente. Si i/d_i y j/d_j son indistinguibles, entonces $F^{i/d_i-j/d_j} = 0$

Para ilustrar la ecuación 12.3, consideremos la figura 12.6:

$$\begin{aligned} F^{1/1-2/1} &= F^{1/1} \oplus F^{2/1} \\ &= \bar{x}_1 x_2 \bar{x}_3 \oplus x_1 \bar{x}_2 \bar{x}_3 \\ &= \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 \end{aligned}$$

Por tanto, las pruebas 010 y 100 pueden distinguir entre las fallas 1/1 y 2/1.

$$\begin{aligned} F^{1/0-2/0} &= F^{1/0} \oplus F^{2/0} \\ &= x_1 x_2 \bar{x}_3 \oplus x_1 x_2 \bar{x}_3 \\ &= 0 \end{aligned}$$

Por tanto, las fallas 1/0 y 2/0 son indistinguibles.

Podemos localizar con precisión las fallas que son distinguibles de las demás utilizando un conjunto completo de prueba para la localización de fallas (FLTS). Sin embargo, en el caso de una falla indistinguible de las demás sólo puede localizarse el conjunto de fallas equivalentes que la contiene. La precisión con que un FLTS puede localizar una falla es la *resolución de fallas* del conjunto.

Un conjunto de prueba es un conjunto de prueba para la localización de fallas si y sólo si la respuesta de la red a la secuencia de entrada de la prueba identifica de manera única la falla con la resolución deseada. Un FDTS debe producir una respuesta que identifique a la red como libre de fallas o que identifique la falla con la resolución deseada. El conjunto {001, 010, 011, 100, 110} es un FDTS para la red de la figura 12.8.

Podemos presentar de manera conveniente el significado de las respuestas que puede producir un conjunto de prueba mediante un *diccionario de fallas*. La tabla 12.7 muestra un diccionario de fallas con máxima resolución para la red de la figura 12.8. Construimos este diccionario determinando las respuestas de la red a cada entrada del conjunto de prueba en presencia de cada condición única de falla simple.

TABLA 12.7 DICCIONARIO DE FALLAS PARA LA FIGURA 12.8

Secuencia de prueba:	001	–	010	–	$x_1x_2x_3$ 011	–	100	–	110	Condiciones posibles
Secuencia de respuesta, z	0		0		0		0		0	2/0,8/0
	0		0		0		0		1	3/0,5/0,7/0
	0		0		1		0		0	1/0,4/0,6/0
	0		0		1		0		1	Sin fallas
	0		0		1		1		1	4/1
	0		1		1		0		1	1/1, 3/1
	1		0		1		0		1	5/1
	1		0		1		1		1	2/1
	1		1		1		1		1	6/1,7/1,8/1

12.3.6 Verificación aleatoria

Los métodos para la generación de pruebas descritos hasta ahora son *deterministas*, ya que elegimos cada vector de prueba en forma explícita para detectar un conjunto específico de fallas. En los circuitos de gran tamaño, la generación de conjuntos de prueba completos deterministas es costosa y, con frecuencia, poco práctica. Si se utilizan métodos integrados de verificación, es difícil construir circuitos que generen en forma automática conjuntos de prueba deterministas.

En cambio, los vectores de prueba generados en forma aleatoria suelen lograr una alta cobertura de fallas (aunque por lo general menor del 100%). La figura 12.14 muestra la relación entre la cobertura de fallas y el número de vectores de prueba aplicados durante la verificación de un circuito típico de gran tamaño. Los primeros vectores de prueba aplicados a un circuito por lo general detectan un alto porcentaje de las fallas posibles. En consecuencia, la cobertura de fallas obtenida mediante un conjunto pequeño de vectores de prueba generados de manera aleatoria puede ser lo bastante alta como para producir un grado aceptable

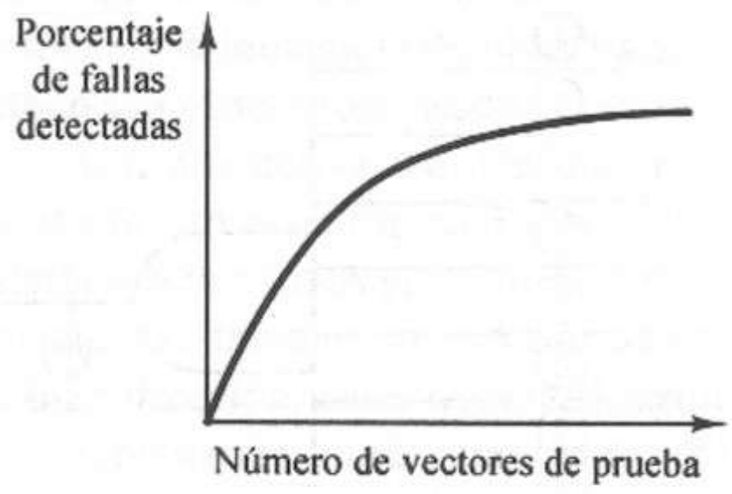


Figura 12.14 Cubierta de fallas contra número de vectores de prueba.

de confianza en el hecho de que el circuito no tiene fallas. Los vectores de prueba aleatorios o seudoaleatorios son generados fácilmente mediante algoritmos implantados en software en un microprocesador de prueba o en circuitos integrados de prueba especiales.

En general, podemos realizar 2^{2^n} funciones diferentes mediante un circuito lógico combinatorio de n entradas, todas las cuales son incorrectas, menos una. Para un vector de entrada dado, la mitad de las funciones posibles producen una salida 0, y la otra mitad produce una salida 1. Así, un único vector de prueba determina si el circuito realiza una de las $(2^{2^n})/2 = 2^{2^n-1}$ funciones con fallas o si es una de las $(2^{2^n})/2$ funciones que tienen la misma salida que la función correcta. Si se obtiene un resultado correcto, podemos concluir que el circuito no realiza una de las 2^{2^n-1} funciones con fallas. Un segundo vector elimina la mitad de las funciones restantes, etcétera. En general, si se obtienen respuestas correctas para m vectores de prueba, la probabilidad de haber detectado todas las fallas en el circuito es aproximadamente

$$P_D = \frac{\sum_{i=1}^{m} 2^{2^n - i}}{2^{2^n} - 1}$$

Al aplicar un número suficientemente grande de vectores de prueba, m, P_D se puede hacer muy cercano a 1. Sin embargo, este valor no es exacto. Muchas fallas se resisten a la verificación mediante patrones aleatorios.

Consideremos la compuerta AND de ocho entradas de la figura 12.15a. La falla 9/1 es detectada por cualquier vector de entrada, excepto $x_1 ... x_8 = \{11111111\}$. Por tanto, la probabilidad de detectar la falla 9/1 mediante un patrón aleatorio es 255/256. En contraste, sólo existe una prueba para la falla 1/1, que es $x_1 ... x_8 = \{01111111\}$, así que, la probabilidad de que la falla 1/1 sea detectada mediante un patrón aleatorio es 1/256.

En general, las compuertas con un *fan–in* de gran tamaño son *resistentes a patrones aleatorios*, pues están sujetas a fallas con baja probabilidad de detección mediante patrones aleatorios. Otras fallas resistentes a patrones aleatorios son aquellas que aparecen en las líneas de señal separadas por muchos niveles de lógica de las entradas primarias o las salidas primarias. En muchos casos, un diseño cuidadoso puede eliminar la mayor parte de las fallas resistentes a patrones aleatorios de un circuito.

Cuando un circuito contiene fallas resistentes a patrones aleatorios, con frecuencia se utilizan pruebas deterministas para detectarlas, y el resto del circuito se verifica mediante una secuencia de patrones aleatorios.

Existen programas para análisis de verificabilidad que evalúan las capacidades de control y observación de fallas en los circuitos digitales. Un

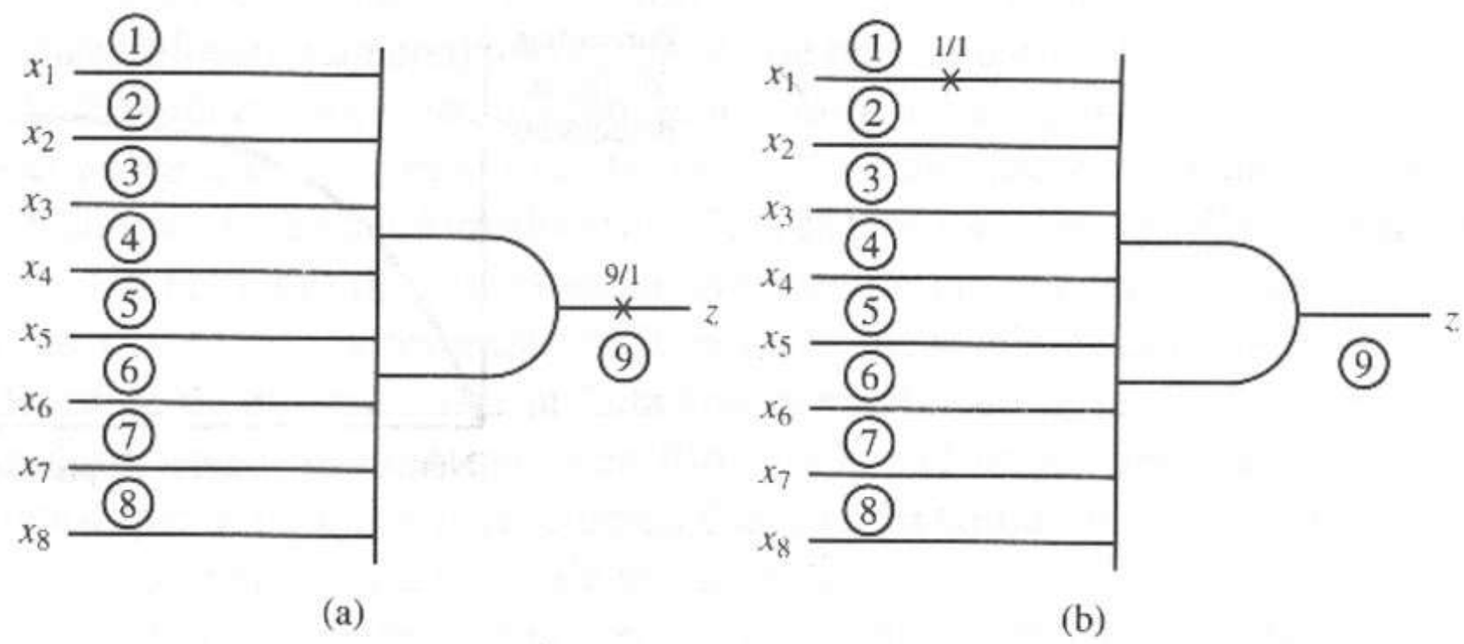


Figura 12.15 Verificación con patrones de prueba. **(a)** Falla 9/1 fácilmente verificable. **(b)** Falla 1/1 resistente a patrones aleatorios.

ejemplo es el Sandia Controllability Observability Analysis Program (SCOAP) [15]. El lector puede consultar [16] si desea más ejemplos.

12.4 Verificación de circuitos lógicos secuenciales

Ahora centraremos nuestra atención en el problema de diagnóstico de fallas de los circuitos lógicos secuenciales. Consideraremos los circuitos secuenciales síncronos que podemos representar mediante el modelo de máquina de estados finitos del capítulo 8. La figura 8.1 muestra un diagrama de bloques del modelo. Repetimos las ecuaciones que describen al modelo:

$$\begin{aligned} z_i &= g_i(x_1, \ldots, x_n, y_1, \ldots, y_r), & i = 1, \ldots, m \\ Y_j &= h_j(x_1, \ldots, x_n, y_1, \ldots, y_r), & j = 1, \ldots, r \\ y_j^{k+1} &= Y_j^k & j = 1, \ldots, r \end{aligned}$$

En el siguiente análisis, las entradas del circuito $x_1,\ldots,x_n$ serán las *entradas primarias* de la lógica combinatoria en el modelo. Las salidas del circuito $z_1,\ldots,z_m$ serán las *salidas primarias*. Los estados $y_1,\ldots,y_r$ y los estados siguientes $Y_1,\ldots,Y_r$ serán las *entradas secundarias* y *salidas secundarias*, respectivamente.

Supondremos que, en general, sólo las entradas primarias se pueden controlar de manera independiente y que durante la verificación sólo se pueden observar las salidas primarias. En otras palabras, las pruebas sólo se pueden aplicar a las entradas $x_1,\ldots,x_n$, y las respuestas de la verificación sólo se pueden observar en las salidas $z_1,\ldots,z_m$. Por tanto, no podemos observar los estados del circuito de manera directa.

También supondremos que el circuito sin fallas es la realización de una tabla de estados reducida. En otras palabras, la tabla de estados correspondiente al circuito sin fallas no tiene fallas equivalentes. Por último, supondremos que el circuito sin fallas es fuertemente conexo. Un circuito es *fuertemente conexo* si y sólo si para cada pareja ordenada de estados (S_i,S_j) del circuito existe una secuencia de entrada que lleva el circuito del estado S_i al estado S_j.

Por lo general, una secuencia de prueba para una falla dada constará de dos subsecuencias ajenas, llamadas *secuencia de inicialización* (IS) y *secuencia*

de observación (OS). La finalidad de la secuencia de inicialización es llevar la máquina en verificación de un estado inicial desconocido a un estado conocido. Entonces, la secuencia de observación hace que el resultado de la transición de estado sea observable desde las salidas primarias.

Una alternativa al uso de una secuencia de inicialización es la incorporación de una lógica *reset* especial al realizar la máquina. Sin embargo, la lógica *reset* estaría sujeta a fallas y, en muchos casos no sería práctica para todos los estados posibles de una máquina. Antes de presentar métodos para obtener secuencias de inicialización, necesitamos el siguiente material básico.

Una *secuencia de transferencia* (TS) para los estados S_i y S_j de una máquina secuencial es la secuencia de entrada más corta que llevará la máquina del estado S_i al estado S_j. El siguiente ejemplo ilustra la deducción de secuencias de entrada.

EJEMPLO 12.2

Deducir la secuencia de entrada mínima que lleva el circuito secuencial descrito por la tabla de estados de la figura 12.16a del estado *A* al estado *B*.

Para lograr esto, suponemos que el circuito se encuentra en el estado A y formamos el árbol de la figura 12.16b. El árbol, obtenido de la tabla de estados, indica que una entrada 0 o 1, aplicada al circuito en el estado A transferirá el circuito al estado A o al estado C, respectivamente. Generamos el árbol completo siguiendo este procedimiento y terminando una rama siempre que un estado se duplica.

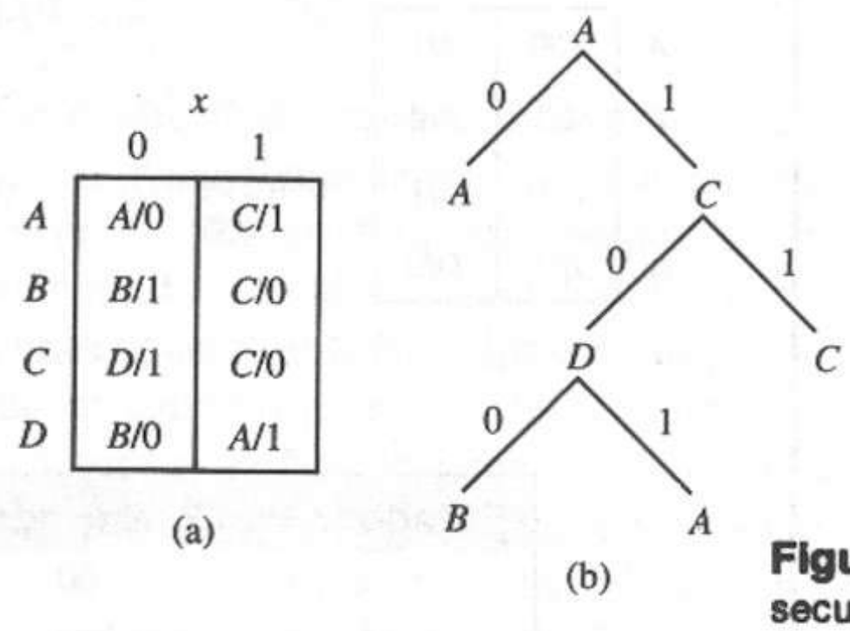


Figura 12.16 Deducción de una secuencia de transferencia. **(a)** Tabla de estados. **(b)** Árbol de transferencia.

La figura 12.16b muestra que la secuencia de transferencia más corta que llevará el circuito del estado A al estado B es $x = 100$.

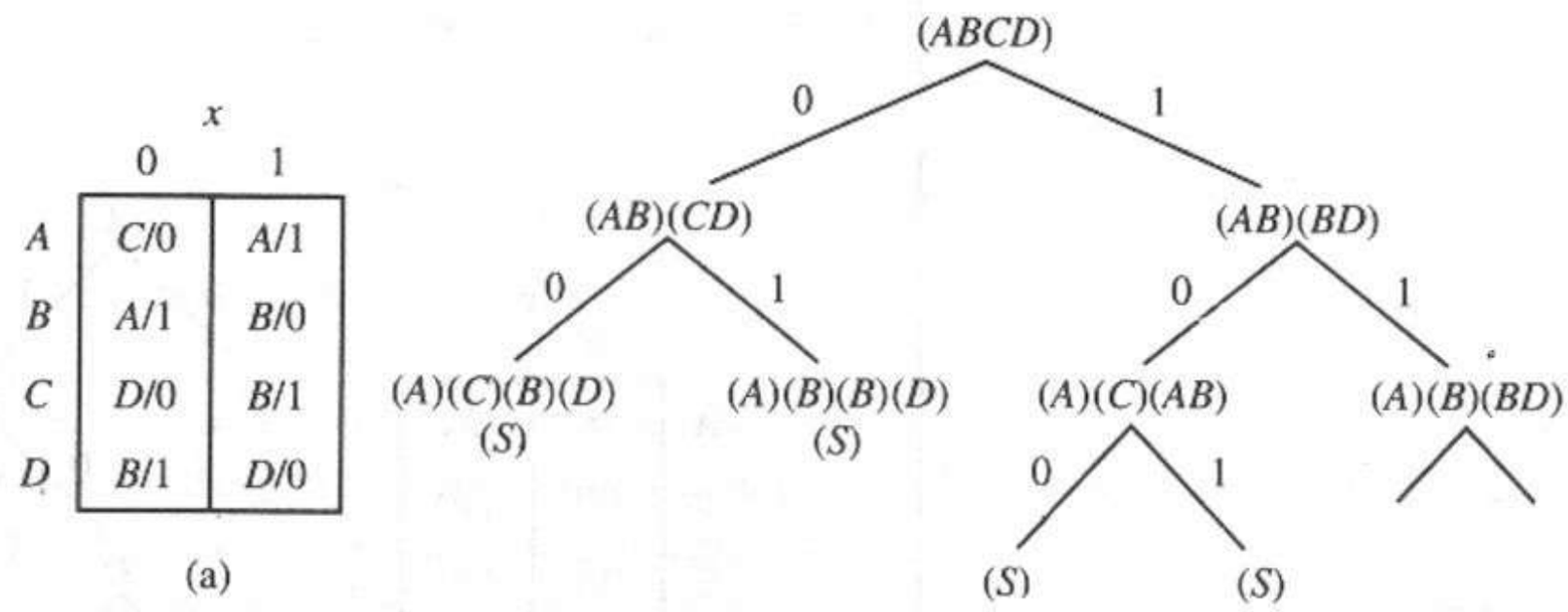
Una *secuencia de retorno* (HS) es una secuencia de entrada que produce una respuesta de salida que identifica de manera única el estado de una máquina después de aplicar la secuencia de retorno. Una *secuencia de retorno con preset* es una secuencia de retorno que no utiliza la respuesta de salida para determinar las entradas posteriores de la secuencia. En otra palabras, los símbolos de la

secuencia son independientes de la respuesta a la secuencia. El siguiente ejemplo muestra la deducción de las secuencias de retorno con *preset*.

EMPLO 12.3

Deducir una secuencia de retorno para el circuito secuencial definido mediante la tabla de estados de la figura 12.17a.

Deducimos la secuencia de retorno mediante el árbol de retorno de la figura 12.17b. Cada nodo del árbol representa un conjunto de estados conocido como *ambigüedad*, pues indica la falta de conocimiento acerca del estado actual del circuito. Se traza una rama desde cada nodo para cada posible entrada al circuito. Construimos el árbol como sigue. Supongamos que tenemos una ambigüedad (*ABCD*), lo que indica que el estado del circuito es desconocido. Determinamos la ambigüedad (*AB*)(*CD*) para una entrada 0. Observe que el componente (*AB*) corresponde a una salida 1, y el componente (*CD*) corresponde a una salida 0. Deducimos la secuencia de retorno completa generando las ramas de esta manera. Una rama termina cuando todos los componentes de una ambigüedad sólo tienen un estado o cuando se repite una ambigüedad. Las ambigüedades que sólo contienen estados simples se rotulan con una (*S*), como en la figura 12.17b. Las secuencias de entrada que van de la ambigüedad inicial (*ABCD*) a una ambigüedad que sólo tiene componentes con estados simples son secuencias de retorno.



Estado inicial	Entrada	Salida	Estado final
A	00	00	*D*
B	00	10	*C*
C	00	01	*B*
D	00	11	*A*

(c)

Figura 12.17 Secuencia de retorno *preset*. **(a)** Tabla de estados. **(b)** Árbol de retorno. **(c)** Secuencia de retorno 00.

La figura 12.17c muestra un análisis de la secuencia de retorno 00. Observe que podemos identificar cada secuencia de salida con un estado final único.

Todas las máquinas secuenciales reducidas tienen al menos una secuencia de retorno. Por definición, una máquina fuertemente conexa tiene al menos una secuencia de transferencia para cada pareja ordenada de estados. Por tanto, podemos inicializar una máquina reducida fuertemente conexa M con cualquier estado S_i mediante el siguiente método:

1. Elegimos una secuencia de retorno (HS) para M.
2. Aplicamos HS a M y observamos la respuesta de salida.
3. Determinamos el estado de M aplicando HS. Llamamos a este estado S_j.
4. Si $S_i \neq S_j$, aplicamos una secuencia de transferencia TS (S_i,S_j).

La generación de secuencias de inicialización recién descrita atañe a los circuitos sin fallas. Por tanto, si aplicamos una secuencia de inicialización determinada de esta manera a un circuito con fallas, la inicialización deseada podría o no ocurrir. Esto sugiere dos preguntas. En primer lugar, si la inicialización deseada no ocurre, ¿podríamos detectar la falla mediante el método aquí descrito? En segundo lugar, ¿podemos generar secuencias de inicialización independientes de la falla? Las respuestas a estas preguntas están más allá del objetivo de este texto.

En la práctica, el diseño de las secuencias de inicialización y observación a menudo no es factible para circuitos con más de unos cuantos estados. En tales casos, se utiliza un diseño especial de métodos de verificabilidad en los circuitos secuenciales para mejorar la capacidad de llevar el circuito a un estado deseado y observar sus salidas. Así, se puede reducir el costo de desarrollo de una prueba para el circuito. Examinaremos estos métodos en la siguiente sección.

12.5 Diseño para la verificabilidad

Se han desarrollado varias técnicas para mejorar la verificabilidad de los circuitos lógicos digitales; es decir, para facilitar la deducción y aplicación de procedimientos de prueba. Un diseño cuidadoso puede hacer que las líneas de señales lógicas de un circuito sean más controlables u observables. La *capacidad de control* de una línea de señal es la facilidad con que puede llevarse a un valor lógico deseado desde las entradas primarias del circuito. La *capacidad de observación* de una línea lógica es la facilidad con que el valor lógico actual de esa línea se puede propagar hasta una salida primaria observable.

Además de mejorar las capacidades de control y observación a través de los métodos de diseño lógico, con frecuencia se integran en el diseño del circuito circuitos especiales que participan en la aplicación de vectores de prueba y la captura de resultados. En muchos casos, estas características especiales de autoverificación integrada (BIST) se crean mediante modificaciones simples del diseño; en otros casos, hay que agregar circuitos exclusivos para verificación al circuito.

En esta sección analizaremos varias técnicas para mejorar la verificabilidad de un circuito secuencial, incluidos el método de diseño de una ruta de rastreo para mejorar las capacidades de control y observación de las variables de estado de los circuitos lógicos secuenciales y las técnicas para crear circuitos BIST que se utilizan en los circuitos integrados y en las tarjetas de circuitos impresos.

12.5.1 Diseño de una ruta de rastreo

Un circuito secuencial con n entradas y r elementos de memoria, como el de la figura 12.18a, requiere 2^{n+r} vectores de prueba para realizar una verificación exhaustiva del bloque de lógica combinatoria (CLB). Sin embargo, las entradas $y_1,...,y_r$ de las variables de estado del CLB, es decir, las salidas del elemento de memoria, no se pueden controlar de manera directa mediante las entradas primarias $x_1,...,x_n$. De manera análoga, las salidas del estado siguiente de CLB, $Y_1,...,Y_r$ no son observables de manera directa a partir de las salidas primarias $z_1,...,z_m$. Por tanto, se necesitan muchos más que 2^{n+r} vectores de prueba para obtener todos los 2^{n+r} patrones en las entradas del CLB y verificar que se generan las salidas adecuadas del CLB.

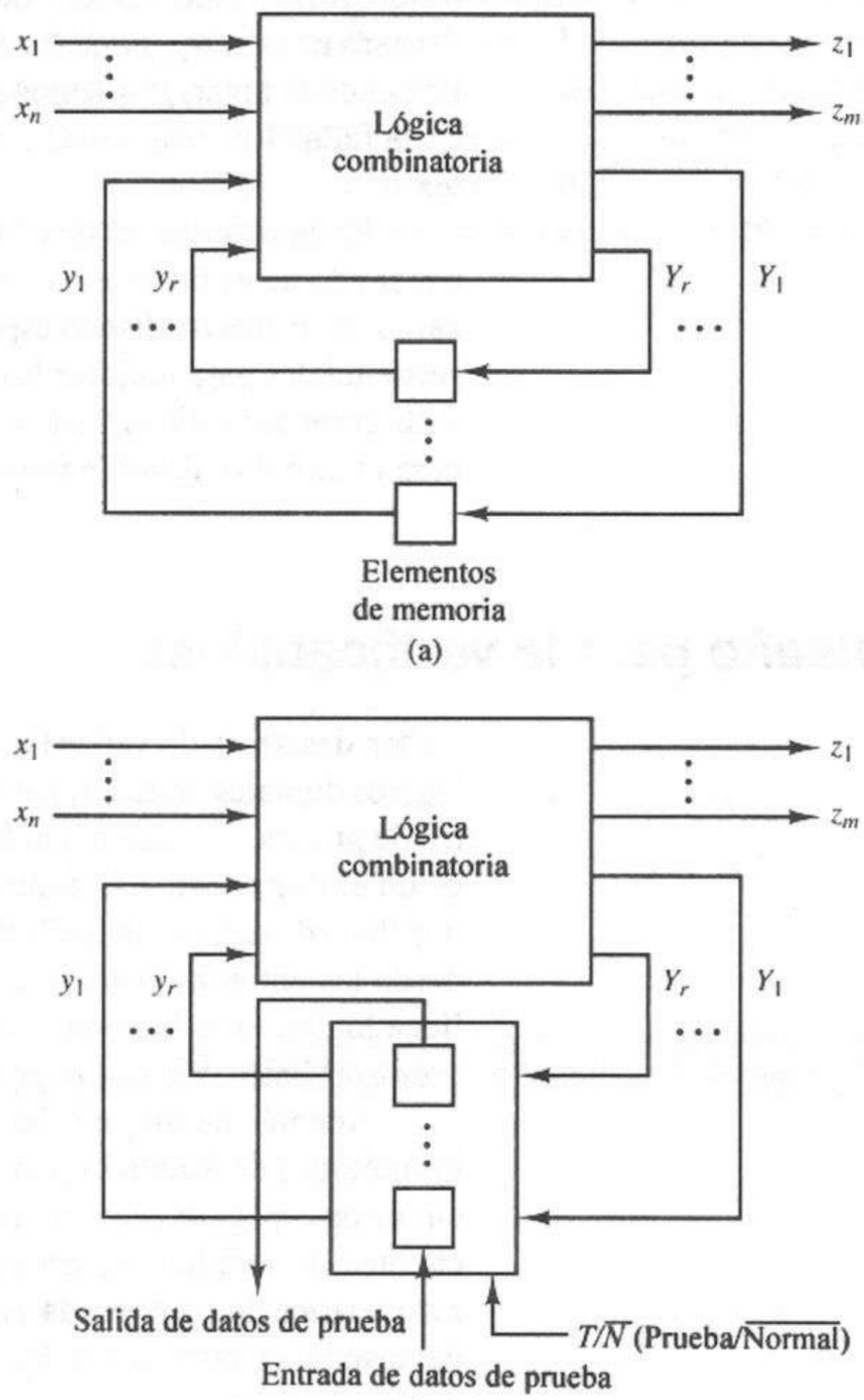


Figura 12.18 Incorporación del diseño de rastreo en un circuito secuencial. **(a)** Modelo genérico de circuito secuencial. **(b)** *Flip-flops* aislados de la lógica combinatoria para su verificación.

Podemos mejorar de manera drástica la verificabilidad de un circuito secuencial aislando los elementos de memoria durante las operaciones de verificación y proporcionando entradas y salidas especiales para controlar y observar sus valores.

El diseño de una ruta de rastreo es un método común para que los *flip-flops* de un circuito secuencial se puedan configurar con los modos de operación normal o para verificación. La figura 12.18b muestra que se añade un *pin* de control especial al circuito, $T/\overline{N}$, para configurar los *flip–flops* con el modo de verificación o el modo normal. Para proporcionar los dos modos de operación, se añade un multiplexor 2 a 1 en cada entrada de excitación del *flip–flop*, como se ve en la figura 12.19 con el multiplexor controlado por el *pin* $T/\overline{N}$. En el *modo normal* ($T/\overline{N} = 0$), las salidas de estado siguiente del CLB $Y_1, \ldots Y_r$ se conectan a las entradas de excitación de los *flip–flops*; es decir, $D_i = Y_i$ para cada *flip–flop*. Así, el circuito opera de acuerdo con su tabla de estados. En el *modo de verificación* ($T/\overline{N} = 1$), cada entrada del *flip–flop* se conecta con la salida del *flip–flop* anterior; es decir, $D_i = y_{i-1}$, configurando los *flip–flops* en un registro de corrimiento serial, llamado *ruta de rastreo*. Observe que la entrada al *flip–flop* y_1 se conecta al *pin Scan–in*, y la salida del *flip–flop* y_r se conecta al *pin Scan–out*.

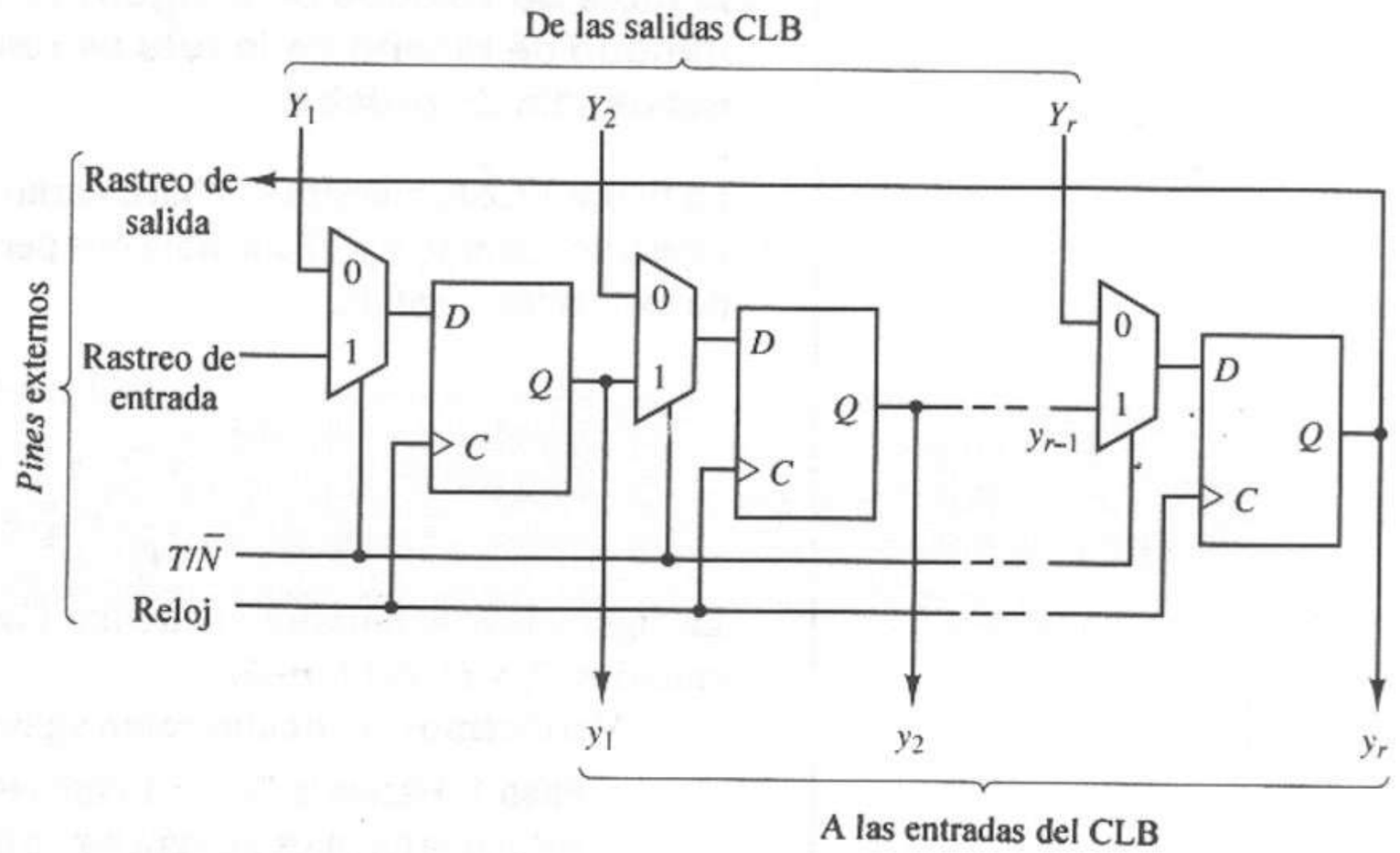


Figura 12.19 Diseño de registro con ruta de rastreo.

El procedimiento de verificación de un circuito secuencial síncrono implantado por el método de diseño con ruta de rastreo es el siguiente:

Paso 1. Verificar la operación de los *flip-flops*, haciendo $T/\overline{N} = 1$ y recorriendo un patrón dado de ceros y unos a través de la ruta de rastreo mediante la línea *Scan-in*. Verificar que la secuencia en *Scan-out* es igual a la secuencia aplicada a *Scan-in*.

Paso 2. Podemos continuar de dos maneras: (a) verificar la tabla de estados, o (b) verificar la presencia de fallas de enclavado en el CLB.

(a) Para verificar la tabla de estados del circuito, realizamos la siguiente prueba del bloque de lógica combinatoria:

1. Hacemos $T/\overline{N} = 1$ para seleccionar el modo de verificación.
2. Recorremos un patrón $y_1,...,y_r$ hacia los *flip-flops* para llevar el circuito a un estado específico.
3. Si se han verificado todas las transiciones de estado de la tabla de estados, terminamos. En caso contrario, aplicamos un patrón de entrada a $x_1,...,x_n$.
4. Observamos las salidas primarias $z_1,...,z_m$ y verificamos que $\mathbf{z} = f(\mathbf{x},\mathbf{y})$.
5. Hacemos $T/\overline{N} = 0$ para elegir el modo normal.
6. Aplicamos reloj al circuito para forzar a una transición de estado, es decir, para cargar $Y_1...Y_r$ en los *flip-flops*.
7. Hacemos $T/\overline{N} = 1$ para seleccionar el modo de prueba.
8. Recorremos el nuevo estado, correspondiente al valor de $Y_1,...,Y_r$ producido por el CLB, y verificamos que $\mathbf{y} = f(\mathbf{x},\mathbf{y})$. Conforme recorremos el estado actual, introducimos al siguiente patrón para $y_1,...,y_r$. Regresar al paso 3.

(b) Aplicamos el FDTS al CLB.

EJEMPLO 12.4

Diseñar un circuito secuencial síncrono para la tabla de estados de la figura 12.20a mediante el método de diseño de la ruta de rastreo, y diseñar una secuencia de prueba.

La figura 12.20b muestra la tabla binaria de estados de una asignación de estados específica para y_1 y y_2. Esta tabla nos permite deducir las siguientes ecuaciones de excitación y salida:

$$D_1 = Y_1 = \bar{x}y_1 + y_1y_2$$
$$D_2 = Y_2 = x\bar{y}_1 + \bar{x}y_1\bar{y}_2 + xy_2$$
$$z = xy_2 + y_1y_2$$

La figura 12.20c muestra el circuito lógico. Observe los multiplexores en las entradas D_1 y D_2 del *flip-flop*.

Verificamos el circuito como sigue:

Paso 1. Hacemos $T/\overline{N} = 1$ para elegir el modo de verificación y recorrer una secuencia de ceros y unos a través de la ruta de rastreo, digamos {010100110}, a fin de verificar que todos los *flip-flops* en la ruta de rastreo pueden alternar entre 0 y 1.

Paso 2. Verificamos la tabla de estados de la figura 12.20a. La figura 12.20d resume este proceso de prueba, el cual verifica cada una de las transiciones de estado y salidas de la tabla de estados. En primer lugar, el circuito es obligado a pasar al estado *A* introduciendo 00 en los *flip-flops*, haciendo $y_1y_2 = 00$ y haciendo $x = 0$. Después de verificar que $z = 0$, el circuito regresa al modo normal haciendo $T/\overline{N} = 0$, y el circuito se programa para que realice una transición de estado. A continuación, se observan las nuevas salidas del *flip-flop* en el siguiente paso de la prueba, regresando el circuito al modo de verificación y recorriendo el contenido de los *flip-flops*, verificando que se ha

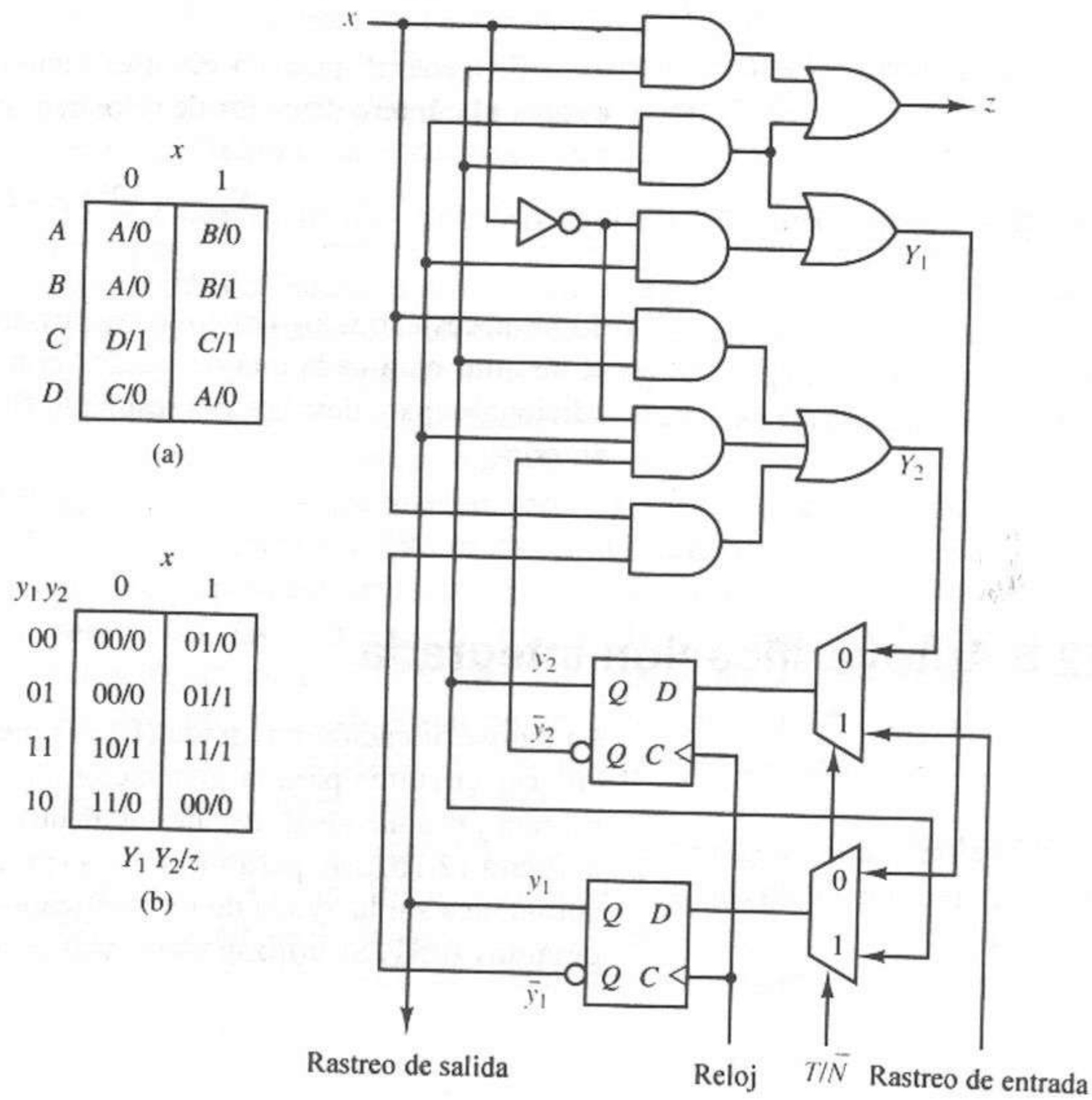
	x = 0	x = 1
A	$A/0$	$B/0$
B	$A/0$	$B/1$
C	$D/1$	$C/1$
D	$C/0$	$A/0$

(a)

$y_1 y_2$	x = 0	x = 1
00	00/0	01/0
01	00/0	01/1
11	10/1	11/1
10	11/0	00/0

$Y_1 Y_2/z$

(b)



(c)

Ciclos de reloj de prueba	Entradas de prueba: Introducir un nuevo estado y_1y_2	Entradas de prueba: Aplicar entrada x	Salidas de prueba: Verificar salida z	Salidas de prueba: Extraer el estado siguiente anterior Y_1Y_2
1–3	00	0	0	—
4–6	00	1	0	00
7–9	01	0	0	01
10–12	01	1	1	00
13–15	11	0	1	01
16–18	11	1	1	10
19–21	10	0	0	11
22–25	10	1	0	11
26–27	—	—	—	00

(d)

Figura 12.20 Diseño de un circuito secuencial síncrono con ruta de rastreo. **(a)** Tabla de estados. **(b)** Tabla de estados binarios. **(c)** Diagrama lógico. **(d)** Secuencia de prueba para la transición de estados.

alcanzado el estado A ($Y_1Y_2 = 00$). Al desplazar este resultado, el estado para la prueba 2 en la tabla de pruebas de la figura 12.20d pasa a los *flip-flops*. Repetimos este proceso hasta verificar las ocho transiciones de estado.

En general, para un circuito secuencial con n entradas y r variables de estado, el número de ciclos de reloj necesarios para verificar la tabla de estados es

$$\begin{aligned} N_{\text{clks}} &= (2^n) \times (2^r) \times (r+1) + r \\ &= (2^{n+r}) \times (r+1) + r \end{aligned} \tag{12.4}$$

donde necesitamos $r + 1$ ciclos para introducir cada vector de estados y programar el circuito para cada una de las 2^{n+r} combinaciones estado/entrada, y r ciclos adicionales para desplazar el resultado final. De la tabla de estados del ejemplo anterior,

$$\begin{aligned} N_{\text{clks}} &= (2^{1+2}) \times (2+1) + 2 \\ &= 27 \text{ ciclos de reloj} \end{aligned}$$

12.6 Autoverificación integrada

La autoverificación integrada (BIST) mejora la verificación de dispositivos al colocar circuitos para la generación de patrones de prueba, así como para la captura y evaluación de respuestas, dentro del circuito por verificar, como muestra la figura 12.1b. Esto permite que gran parte de la verificación proceda de manera automática sin la ayuda de un verificador. La figura 12.21 ilustra el uso de los circuitos BIST. Se utilizan elementos de un registro de corrimiento con retroali-

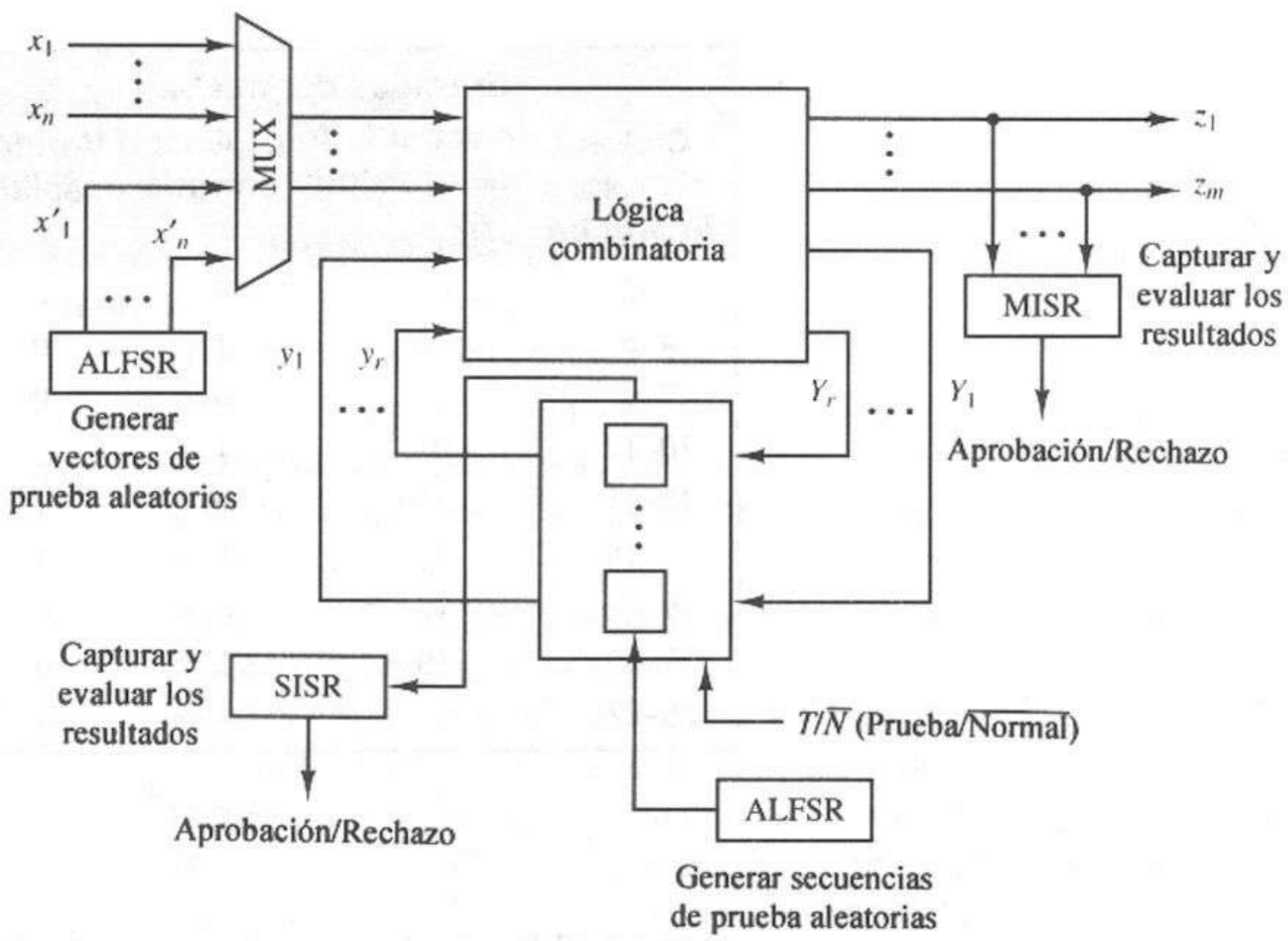


Figura 12.21 Circuito lógico secuencial con autoverificación integrada (BIST).

mentación lineal autónoma (ALFSR) para generar patrones de prueba seudoaleatorios que son aplicados al bloque de lógica combinatoria y a la entrada de la ruta de rastreo. Los datos de las salidas primarias se capturan y comprimen mediante un registro de identificación para varias entradas (MISR), mientras que los datos seriales de la ruta de rastreo son capturados y comprimidos mediante un registro de corrimiento con entrada serial (SISR). En esta sección describiremos el diseño y funcionamiento de cada uno de estos elementos BIST.

12.6.1 Generación seudoaleatoria de vectores de prueba

El circuito BIST más común utilizado para generar vectores de prueba seudoaleatorios es el registro de corrimiento con retroalimentación lineal (LFSR). Un LFSR es una configuración en serie de *flip–flops* D y compuertas OR exclusivo (XOR) cuyo diseño y funcionamiento se basan en los principios de la aritmética polinomial de la teoría de codificación cíclica.

Cuando hay que generar vectores de prueba, con frecuencia se utiliza un *LFSR autónomo* (ALFSR), es decir, un LFSR sin entradas externas. Un ALFSR de n etapas produce una secuencia periódica seudoaleatoria de n números binarios según una *función generatriz* especial realizada mediante las líneas de retroalimentación y compuertas XOR dentro del ALFSR. Si la secuencia tiene $2^n - 1$ valores no nulos, lo que produce un periodo de $2^n - 1$, la función generatriz es un *polinomio primitivo*. (Se evita que ocurra el patrón 0000, pues el LFSR nunca saldría de este estado.)

La figura 12.22a presenta una estructura general utilizada en los diseños ALFSR. En este circuito, $a_{n-1}...a_0$ son las salidas de los n *flip–flops* del registro de corrimiento de n bits, de modo que a_n es la entrada del registro de corrimiento, igual al OR exclusivo de las señales de retroalimentación; es decir,

$$\begin{aligned} a_n &= \sum_{i=0^{\oplus}}^{n-1} a_i c_i \\ &= a_0 c_0 \oplus a_1 c_1 \oplus \cdots \oplus a_{n-1} c_{n-1} \end{aligned}$$

Elegimos los coeficientes $c_{n-1}...c_0$ para crear el polinomio primitivo $p(x)$, de modo que $c_i = 1$ si la salida a_i del *flip–flop* se retroalimenta a la entrada del registro de corrimiento a través de las compuertas OR exclusivo, y $c_i = 0$ si a_i no se conecta al circuito de retroalimentación.

La figura 12.22b presenta un ALFSR de 4 bits basado en el polinomio primitivo

$$p(x) = x^4 + x + 1$$

En la salida a_0, el valor a_1 está presente después de un periodo de reloj, a_2 después de dos periodos de reloj, etcétera. Con frecuencia, esta secuencia de salidas se representa en forma polinomial como

$$f(x) = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

donde x^k es un retardo de k periodos de reloj. Observe que

$$a_4 = a_1 \oplus a_0$$

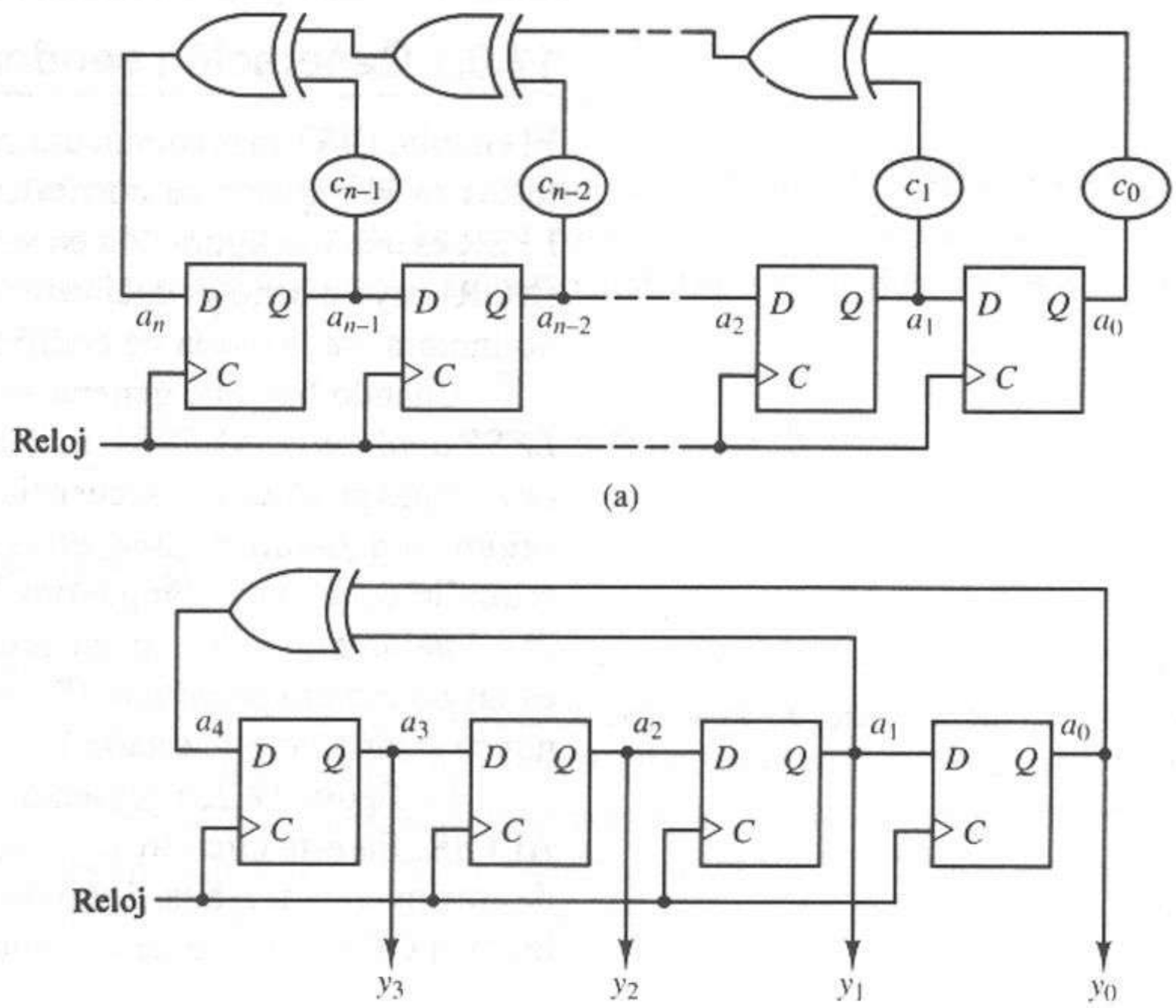
y

$$a_4 \oplus a_1 \oplus a_0 = 0$$

lo cual corresponde al polinomio primitivo

$$\begin{aligned} p(x) &= x^4 + x^1 + x^0 \\ &= x^4 + x + 1 \end{aligned}$$

Si al registro se asigna un estado inicial de 1000, obtenemos la secuencia de la figura 12.22c.



Reloj	a_3	a_2	a_1	a_0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	1	0	0	1
5	1	1	0	0
6	0	1	1	0
7	1	0	1	1
8	0	1	0	1
9	1	0	1	0
10	1	1	0	1
11	1	1	1	0
12	1	1	1	1
13	0	1	1	1
14	0	0	1	1
15	0	0	0	1
16	1	0	0	0

(c)

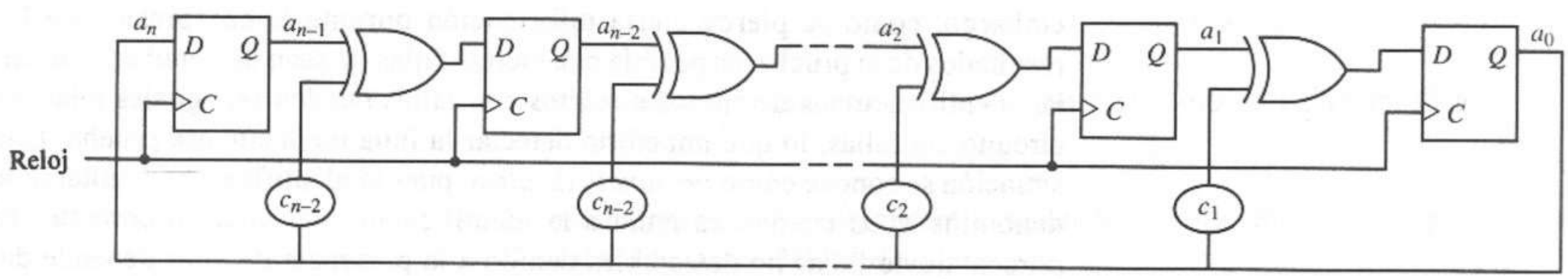
Figura 12.22 Estructura 1 de un ALFSR. **(a)** Estructura general. **(b)** ALFSR con $p(x) = x^4 + x + 1$. **(c)** Secuencia de números generada.

Los polinomios primitivos para secuencias de longitud $2^n - 1$ aparecen publicadas en varias fuentes [13].

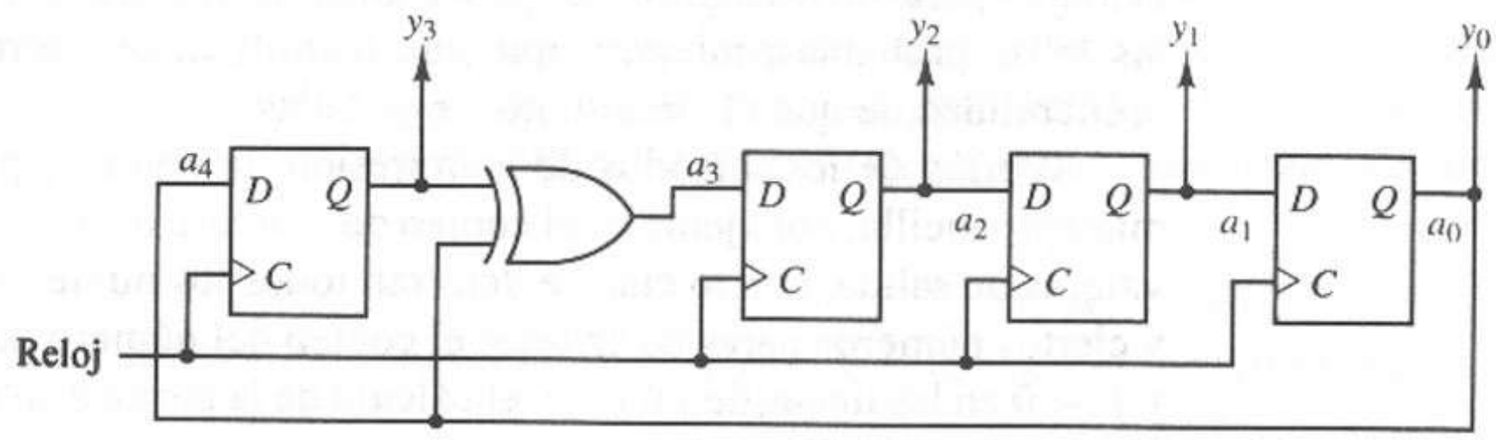
La figura 12.23a muestra un segundo tipo de estructura ALFSR, que se utiliza por lo general cuando existen dos o más derivaciones para retroalimentación. Como en la primera estructura, los valores c_i indican la presencia o ausencia de derivaciones para retroalimentación. La figura 12.23b muestra el ALFSR de 4 bits que utiliza esta estructura y produce la secuencia de conteo de la figura 12.23c. Su polinomio primitivo es

$$p(x) = x^4 + x^3 + x^0$$

Este polinomio se obtiene del de la figura 12.22b reemplazando cada elemento x^j del polinomio primitivo por x^{n-j}. En esta configuración, se coloca una compuerta XOR en la entrada de cada *flip–flop* con el que exista una conexión de retroalimentación. Como resultado, a lo más ocurre un retardo de compuerta



(a)



(b)

Reloj	a_3	a_2	a_1	a_0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	1	0	0
6	0	1	1	0
7	0	0	1	1
8	1	1	0	1
9	1	0	1	0
10	0	1	0	1
11	1	1	1	0
12	0	1	1	1
13	1	1	1	1
14	1	0	1	1
15	1	0	0	1
16	1	0	0	0

(c)

Figura 12.23 Estructura 2 de un ALFSR. **(a)** Estructura general. **(b)** ALFSR con $p(x) = x^4 + x^3 + 1$. **(c)** Secuencia de números generada.

XOR entre los ciclos de reloj, mientras que en la primera estructura todas las compuertas XOR están contenidas en la ruta de retroalimentación, lo que produce $k - 1$ retardos de compuerta para k líneas de retroalimentación. Observe que podemos construir la primera estructura mediante un registro de corrimiento estándar con compuertas XOR añadidas en forma externa, mientras que la segunda estructura requiere la inserción de compuertas XOR entre los *flip-flops*.

12.6.2 Análisis de identificaciones

Con frecuencia, la captura y análisis de las respuestas de un circuito a cada vector de prueba individual de un FDTS no es práctica, en particular si los circuitos de prueba se integran al módulo del circuito. En tales casos, lo usual es comprimir los datos de respuesta a toda una secuencia de pruebas en un único valor de datos llamado *identificación*. Si la identificación de un circuito obtenida para cierta prueba es incorrecta, sabemos que el circuito tiene fallas. Si la identificación es correcta, es más probable que el circuito no tenga fallas. Sin embargo, como se pierde cierta información durante la compresión de los resultados de la prueba, es posible que ciertas fallas no sean detectadas; es decir, las identificaciones de ciertos circuitos con fallas pueden ser iguales a las del circuito sin fallas, lo que impediría detectar la falla mediante esa prueba. Esta situación se conoce como *presencia de alias*, pues la identificación resultante se denomina *alias* cuando es igual a la identificación del circuito correcto. El porcentaje de fallas no detectables debido a la presencia de alias depende del diseño del circuito y del algoritmo utilizado para comprimir los datos. En general, la mayor parte de los algoritmos para análisis de identificación detecta casi todas las fallas probables, mientras que una identificación correcta indica una alta probabilidad de que el circuito no tenga fallas.

Varios de los métodos de compresión de datos se pueden implantar de manera sencilla; por ejemplo, el conteo del número de unos en una secuencia de valores de salida, con lo cual se detectan todos los números impares de errores y ciertos números pares de errores; el conteo del número de transiciones $0 \rightarrow 1$ y $1 \rightarrow 0$ en las líneas de salida; y el cálculo de la paridad sobre una secuencia de salidas, con lo que se detectan todos los errores simples y todos los números impares de errores. Estos métodos se implantan con facilidad, aunque algunos patrones de error de salida tal vez no se detecten debido a la presencia de alias.

Se puede lograr una mayor cobertura de las fallas a través de los métodos de codificación con detección de errores, los cuales se implantan con frecuencia en un LFSR. Supongamos que podemos representar una secuencia de respuesta a una prueba en una línea de salida z mediante un polinomio $z(x)$:

$$z(x) = z_n x^n + z_{n-1} x^{n-1} + \cdots + z_1 x^1 + z_0$$

donde x^k representa un retardo de tiempo de k ciclos de reloj y z^k representa los valores dato en cada ciclo de reloj k, producidos por el $k^{\text{-ésimo}}$ vector de prueba. Recuerde, de nuestro análisis anterior, que un LFSR de k bits representa un polinomio generador de grado k. En la teoría de codificación cíclica, la detección de errores se realiza dividiendo el polinomio de la secuencia de datos $z(x)$ entre el polinomio generador del LFSR, $p(x)$, obteniendo un cociente $q(x)$ y un residuo $r(x)$. Por tanto,

$$z(x) = q(x)p(x) + r(x) \tag{12.5}$$

El LFSR realiza la división de polinomios conforme llega la secuencia de datos de respuesta. Al realizar la operación, los $n - k$ bits del cociente se

eliminan del LFSR, dejando el residuo $r(x)$ de k bits en el LFSR después del último paso.

Si una falla de circuito produce uno o más errores en la secuencia de salidas, podemos representar la secuencia de salida con z^*, donde

$$z^*(x) = z(x) \oplus e(x) \tag{12.6}$$

En la ecuación 12.6, el polinomio $e(x)$ representa una secuencia de error:

$$e(x) = e_n x^n + e_{n-1} x^{n-1} + \cdots + e_1 x^1 + e_0$$

donde $e_k = 1$ si el bit z_k es incorrecto y $e_k = 0$ si el bit z_k es correcto. Observe que $z_k \oplus e_k = \bar{z}_k$ si hay un error, y $z_k \oplus e_k = z_n$ si no existe error. Así, podemos representar la secuencia de salida $z^*(x)$ como

$$z^*(x) = z(x) \oplus e(x) \tag{12.7}$$

$$= (e_n \oplus z_n)x^n + (e_{n-1} \oplus z_{n-1})x^{n-1} + \cdots + (e_1 \oplus z_1)x^1 + (e_0 \oplus z_0) \tag{12.8}$$

La división de polinomios realizada sobre $z^*(x)$ por el LFSR implica

$$z^*(x) = q^*(x)p(x) + r^*(x) \tag{12.9}$$

donde el residuo $r^*(x)$ es la identificación del circuito. Suponemos que el circuito no tiene fallas si $r^*(x) = r(x)$.

Observe que para un cociente de $(n-k)$ bits existen 2^{n-k} cocientes diferentes de entre 2^n secuencias de respuestas posibles con el mismo residuo de k bits $r(x)$. Sólo una de estas secuencias de prueba y cocientes corresponde al funcionamiento sin fallas. Por tanto, la probabilidad de no detectar un error, es decir, la probabilidad de que $r^*(x) = r(x)$ para una secuencia incorrecta $z^*(x)$, es

$$P_M = \frac{2^{n-k} - 1}{2^n - 1} \tag{12.10}$$

Para valores grandes de n, la ecuación 12.10 se reduce a

$$P_M \approx \frac{2^{n-k}}{2^n} = 2^{-k} \tag{12.11}$$

que sólo depende de la longitud del LFSR. Por tanto, si las secuencias de respuesta son largas, la probabilidad de no detectar un error se reduce conforme aumenta el número de etapas del LFSR.

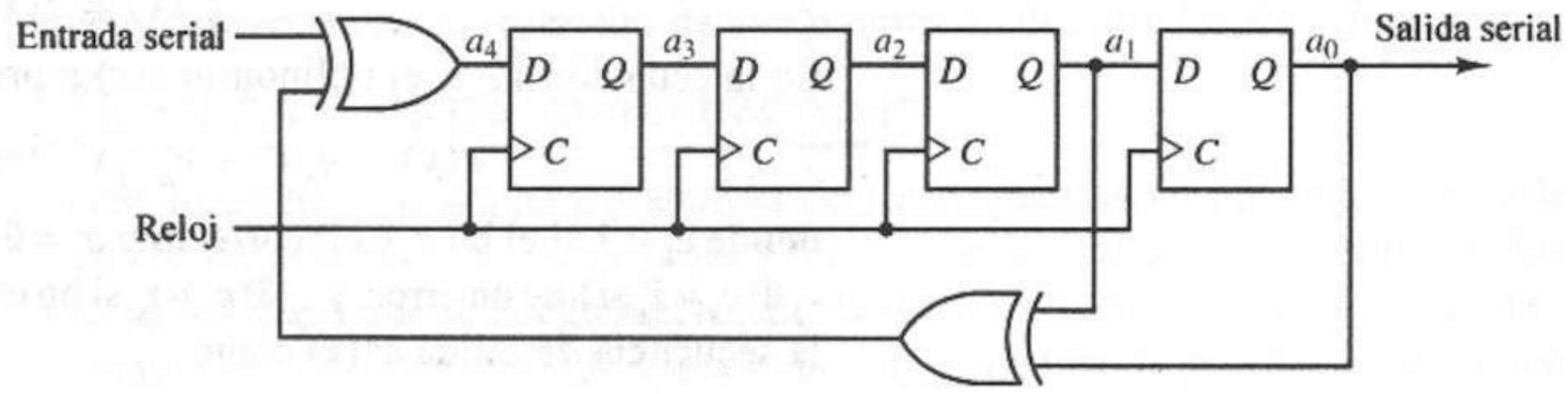
La figura 12.24a muestra un LFSR con una entrada simple; esta configuración es un *registro de identificación de entrada serial* (SISR). Realizamos la función generatriz de un SISR como lo hicimos en el caso de los elementos ALFSR. Para el SISR en la figura 12.24a, la función generatriz es

$$p(x) = x^4 + x + 1$$

ya que las derivaciones de retroalimentación se encuentran en las etapas 1 y 0. La figura 12.24b ilustra el funcionamiento del SISR para la secuencia de entrada $z(x) = 010001101110$. La figura 12.24c muestra la respuesta obtenida cuando ocurre una secuencia de entrada incorrecta; en este caso, hemos alterado el bit 1 de la secuencia de entrada de la figura 12.24c. Observe que la respuesta de error, $r^*(x)$, difiere de la respuesta esperada, $r(x)$.

Cuando un circuito lógico tiene m salidas paralelas, $z_1 \ldots z_m$, podemos utilizar m elementos SISR para calcular las identificaciones individuales de cada salida. Sin embargo, este método es costoso. Un método más eficiente utiliza un *registro*

de *identificación de varias entradas* (MISR), el cual de hecho calcula de manera concurrente las identificaciones de las secuencias de entrada paralelas dentro de un único circuito LFSR. La figura 12.25 muestra un MISR de 4 bits basado en la misma función generatriz del SISR de la figura 12.24a.



(a)

Entrada serial $z(x)$	Contenido del LFSR a_3	a_2	a_1	a_0
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
1	1	0	0	1
1	0	1	0	0
0	1	0	1	0
1	1	1	0	1
1	0	1	1	0
1	0	0	1	1
0	1	0	0	1
$r(x) =$	1	1	0	0

(b)

Entrada serial $z(x)$	Contenido del LFSR a_3	a_2	a_1	a_0
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
1*	0	0	1	0
1	0	0	0	1
1	0	0	0	0
0	1	0	0	0
1	0	1	0	0
1	1	0	1	0
1	0	1	0	1
0	0	0	1	0
$r^*(x) =$	1	0	0	1

(c)

Figura 12.24 Funcionamiento de un SISR. **(a)** Diagrama lógico del SISR. **(b)** Respuesta a una secuencia correcta. **(c)** Respuesta a una secuencia incorrecta.

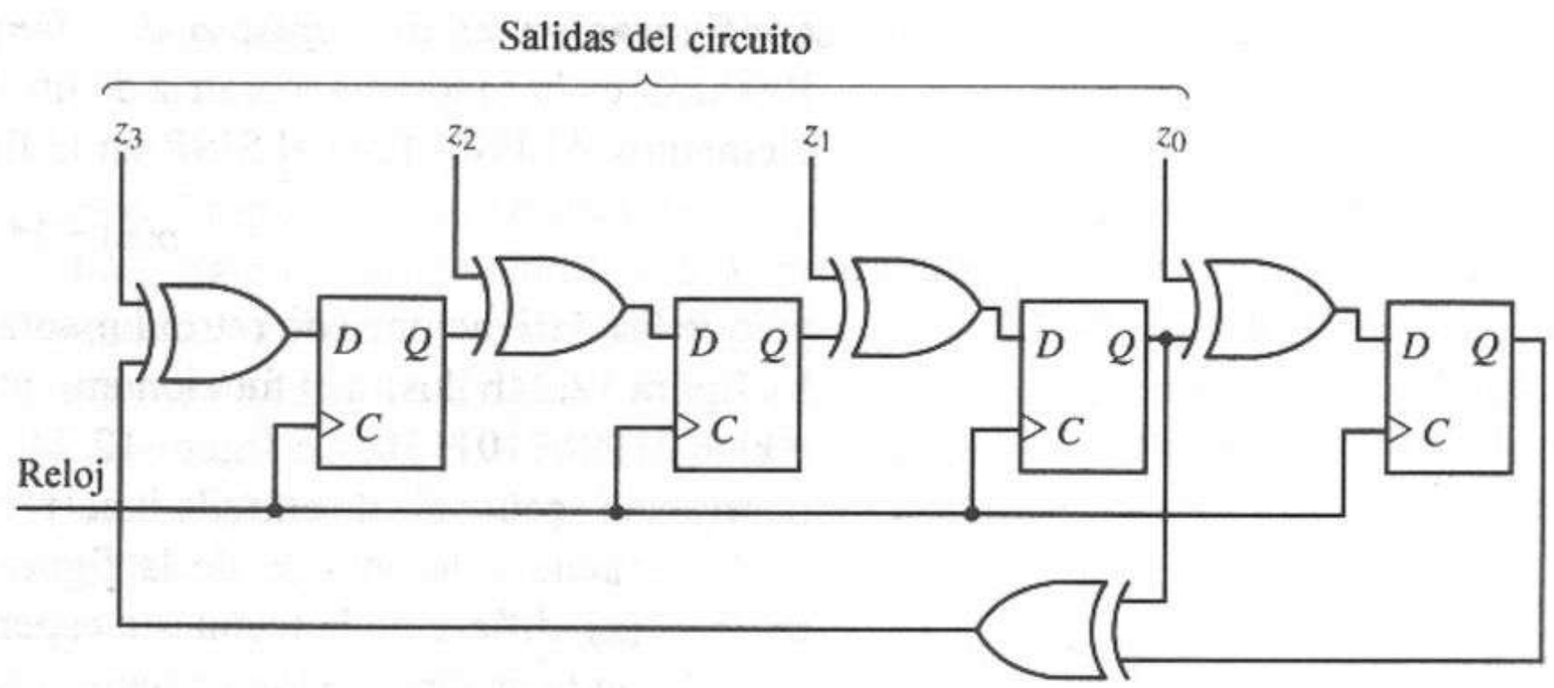


Figura 12.25 Registro de identificación con varias entradas (MISR).

En este caso, la probabilidad de no detectar un error en una de las m secuencias de datos, donde las secuencias de respuesta contienen n valores de m bits, está dada por

$$P_M = \frac{2^{nm-k} - 1}{2^{nm} - 1} \tag{12.12}$$

Como en el caso del SISR, para secuencias largas (es decir, para nm grande),

$$P_M \approx \frac{2^{nm-k}}{2^{nm}} = 2^{-k} \tag{12.13}$$

Por tanto, la probabilidad de no detectar un error en una secuencia grande de valores de datos depende principalmente del número de etapas del MISR. Observe que el número de entradas paralelas al MISR, m, puede ser menor que el número de etapas del MISR, k. En este caso, las entradas se combinan con m de las k entradas de *flip–flop*.

En general, un SISR o MISR de longitud $k \geq 2$ puede detectar todos los errores de un solo bit. Si el polinomio generador del LFSR $p(x)$ tiene un número par de términos [lo que ocurre cuando $p(x)$ es divisible entre $(x + 1)$], podemos detectar cualquier cantidad impar de errores. Además, podemos detectar cualquier *secuencia de errores;* es decir, cualquier grupo de errores consecutivos, hasta de longitud k. Por tanto, la cubierta de la falla es función del número de etapas k del LFSR, del número de entradas paralelas m y de la longitud de las secuencias de entrada n, así como de la función generatriz $p(x)$ y el estado inicial del LFSR. [20] muestra una descripción más detallada de las propiedades de la cubierta de fallas del análisis de identificación.

12.6.3 Observador integrado de bloque lógico

Al agregar elementos del LFSR a un circuito para realizar la generación de patrones y el análisis de identificación se añade a la complejidad y costo a un módulo de circuito. Un método que puede minimizar el número total de *flip–flops* en el circuito consiste en combinar las funciones de generación de patrones y de análisis de identificación con el registro de estado normal del circuito. La estructura de registro resultante es un *observador de bloque lógico integrado* (BILBO).

La figura 12.26 muestra un registro BILBO de 4 bits. El BILBO tiene dos líneas de control, B_1 y B_2, las cuales seleccionan el modo de operación como muestra la tabla 12.8. Para $B_1B_2 = 00$, los *flip–flops* se cargan con las líneas de entrada paralela I_1 a I_4, operando como un registro de carga paralela normal. Cuando $B_1B_2 = 01$ y $B_1B_2 = 11$, cada *flip–flop* (excepto el primero) se carga con la salida del *flip–flop* a su izquierda; es decir, $D_i = F_{i-1}$. Cuando $B_1B_2 = 01$, $D_1 =$ rastreo, y entonces el BILBO se configura como un registro de corrimiento serial que se utilizará como ruta de rastreo. Para $B_1B_2 = 11$, D_1 se conecta a la señal de retroalimentación, configurando al BILBO como un ALFSR. Cuando $B_1B_2 =$ 10, cada entrada de *flip–flop* $D_i = I_i \oplus F_{i-1}$, y entonces el LFSR funciona como un MISR, suponiendo que I_1 a I_4 son salidas del circuito.

Durante el funcionamiento normal del circuito, se elige el modo normal (carga paralela); las entradas del BILBO son las variables de estado siguiente $Y_1 \ldots Y_4$ y las salidas son el estado actual $y_1 \ldots y_4$. Durante las operaciones de verificación, el BILBO se configura en el modo de registro de corrimiento para revisar los valores de entrada y salida. Realizamos la generación de patrones de prueba y el análisis de identificación estableciendo el modo del BILBO como un ALFSR para generar patrones y como MISR para capturar resultados y realizar el análisis de identificación.

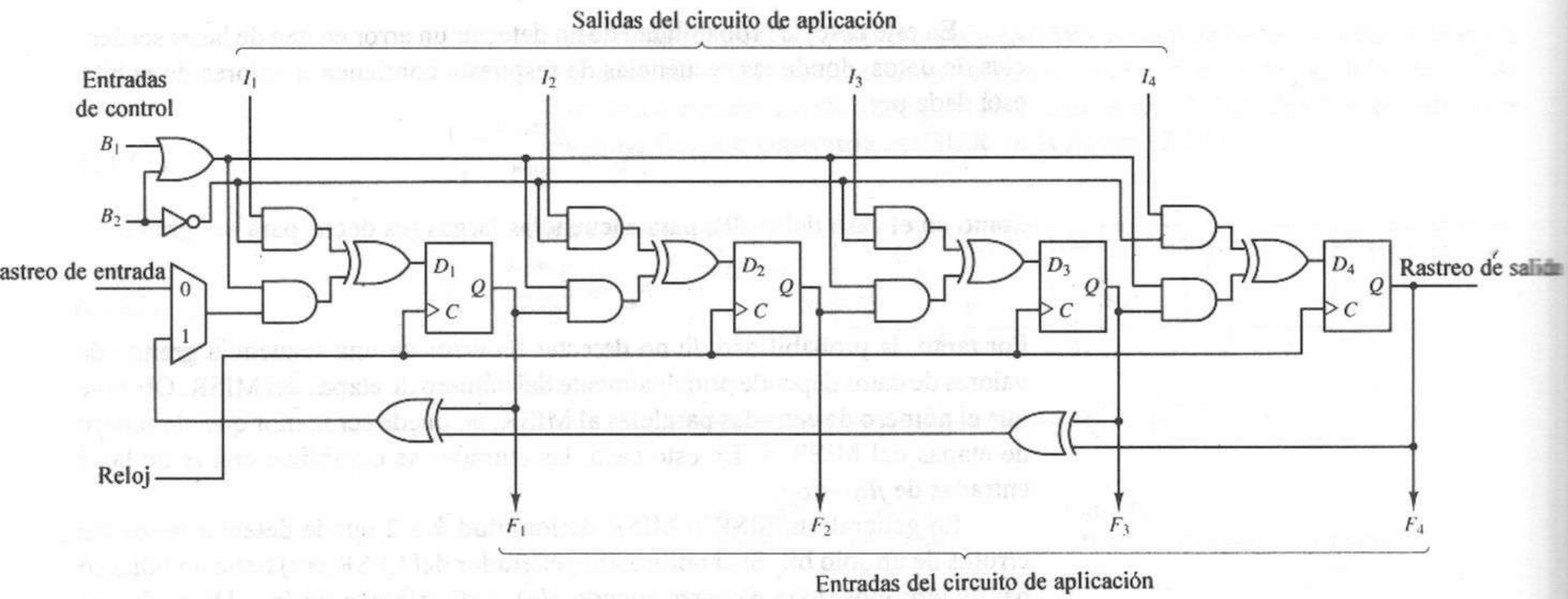


Figura 12.26 Observador integrado de bloque lógico (BILBO).

TABLA 12.8 MODOS DE OPERACIÓN DEL BILBO

B_1	B_2	Entrada del MUX	D_i	Función	Función de prueba
0	0	Rastreo de entrada	I_i	Registro de carga paralela	Modo normal (sin verificación)
0	1	Rastreo de entrada	F_{i-1}	Registro de corrimiento lineal	Modo de ruta de rastreo
1	0	Retroalimentación	$I_i \oplus F_{i-1}$	MISR	Análisis de identificación
1	1	Retroalimentación	F_{i-1}	ALFSR	Generación de patrones

Al utilizar los BILBO, lo usual es dividir el circuito en módulos, como muestra la figura 12.27, permitiendo que los BILBO elegidos funcionen como elementos ALFSR y que los otros operen como elementos MISR durante la verificación. Para el circuito de la figura 12.27, el procedimiento de prueba sería como sigue:

Paso 1. Utilizamos BILBO 1 como ALFSR y BILBO 2 como MISR para probar el CLB 1 de la manera siguiente:

1. Colocamos ambos BILBO en el modo de ruta de rastreo e introducimos los valores iniciales: el valor original del ALFSR en el BILBO 1 y ceros para BILBO 2.
2. Colocamos el BILBO 1 en el modo ALFSR y el BILBO 2 en el modo MISR.
3. Operamos el circuito durante el número indicado de ciclos de prueba. El BILBO 1 genera patrones para las entradas del CLB 1 y el BILBO 2 realiza un análisis de identificación sobre las salidas de CLB 1.

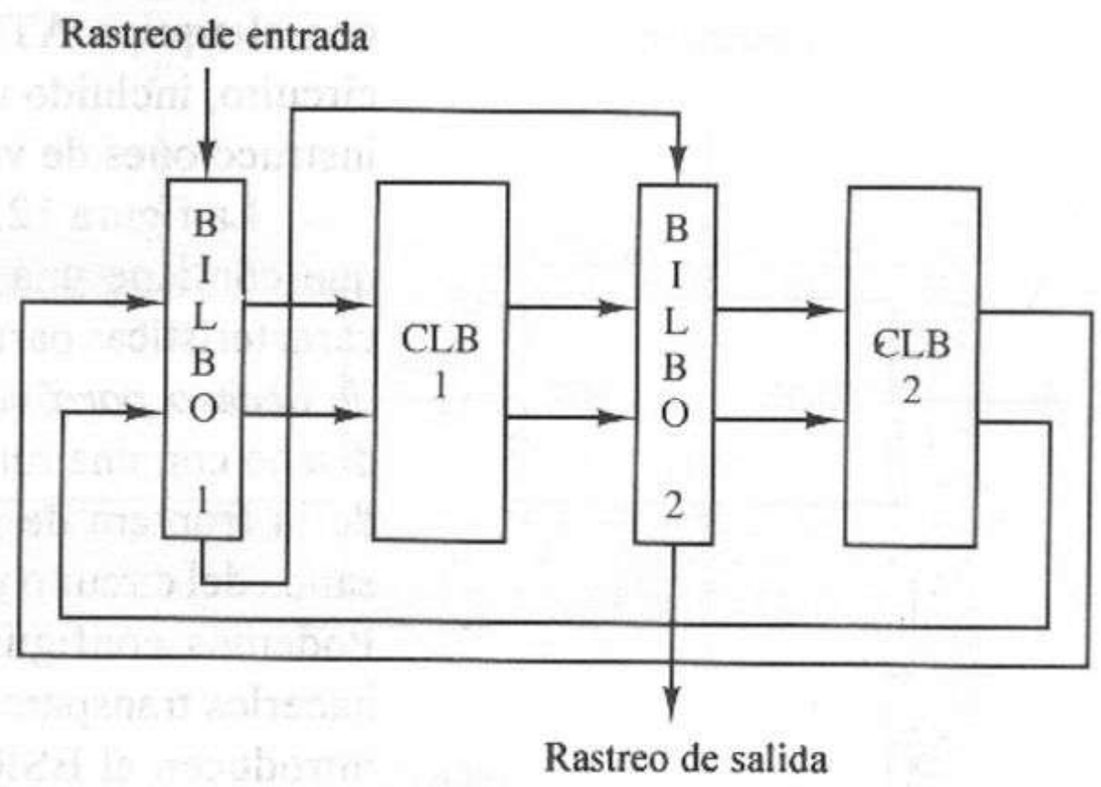


Figura 12.27 División del circuito para su verificación con BILBO.

4. Colocamos ambos BILBO en el modo de ruta de rastreo y extraemos la identificación final del BILBO 2 mientras introducimos los valores iniciales para la siguiente verificación.

Paso 2. Utilizamos el BILBO 2 como ALFSR y el BILBO 1 como MISR para verificar el CLB 2. Realizamos los pasos 2 a 4 del paso 1, intercambiando los papeles de los dos BILBO.

Al terminar la prueba, los BILBO regresan al modo de carga paralela para que puedan operar como *flip–flops* de las variables de estado normal.

12.7 Rastreo de frontera en el nivel de sistema y de tarjeta

Las tarjetas y sistemas de circuitos lógicos digitales representan un reto mayor para los ingenieros de verificación y diseñadores digitales que los circuitos integrados y módulos individuales, aunque gran parte de los conceptos de verificación y diseño verificable analizados en este capítulo se pueden extender a las tarjetas y sistemas. Para enfrentar este problema, varios fabricantes han conformado el Joint Test Advisory Group (JTAG) y en 1988 desarrollaron la JTAG Testability Bus Specification, adoptada en 1990 como estándar 1149.1 por el IEEE [18]. Los objetivos del *bus* de verificabilidad 1149.1 son los siguientes:

1. Una interfaz estándar entre los sistemas ATE y los dispositivos de una tarjeta de circuitos impresos (PCB), que permite transferir los datos de prueba a los dispositivos y recibir de éstos la información de diagnóstico.

2. Un método para verificar las interconexiones entre los circuitos de una PCB, que son una fuente común de fallas en las PCB.

3. Un método para verificar y localizar los circuitos individuales con fallas en una PCB.

Estos objetivos se logran extendiendo los métodos de diseño con rutas de rastreo a toda la tarjeta de circuitos. El bus de verificabilidad consta principalmente de cuatro líneas de señal exclusivas en una PCB, con una interconexión estándar

con el equipo ATE y una interfaz especial del *bus* de verificabilidad en cada circuito, incluido un circuito integrado de control para interpretar y ejecutar las instrucciones de verificación emitidas por el sistema ATE.

La figura 12.28 muestra la configuración básica de un módulo de circuito que contiene una interfaz de *bus* de verificabilidad. Cada módulo tiene dos características particulares: un *registro de rastreo de frontera* (BSR) y un *puerto de acceso para verificación* (TAP). El BSR es una extensión del método de diseño con una ruta de rastreo; proporciona un registro de corrimiento en torno de la frontera de cada circuito, con un *flip–flop* entre cada línea de entrada/salida del circuito y el *pin* externo correspondiente, como muestra la figura 12.29. Podemos configurar estos *flip–flops* como registros de corrimiento serial o hacerlos transparentes para el funcionamiento normal del circuito. Los datos se introducen al BSR a través del *pin* TDI (*test data pin*) exclusivo y se extraen mediante el *pin* TDO (*test data out*).

Los BSR de los componentes individuales de una tarjeta de circuitos se configuran en un único registro de corrimiento serial conectando el *pin* TDO de un dispositivo con el *pin* TDI del siguiente, de modo que un *pin* TDI y uno TDO se conectan al sistema ATE. La figura 12.30 ilustra esto. Con este registro de corrimiento, el sistema ATE puede proporcionar instrucciones y datos de prueba a todos los circuitos de la tarjeta y recibir los resultados de la verificación y otros datos de diagnóstico de ellos. Si se desea, se puede eliminar cualquier circuito individual del registro de corrimiento mediante un *registro de desvío* de 1 bit, el cual aparece en la figura 12.28 y dirige los datos recibidos del *pin* TDI

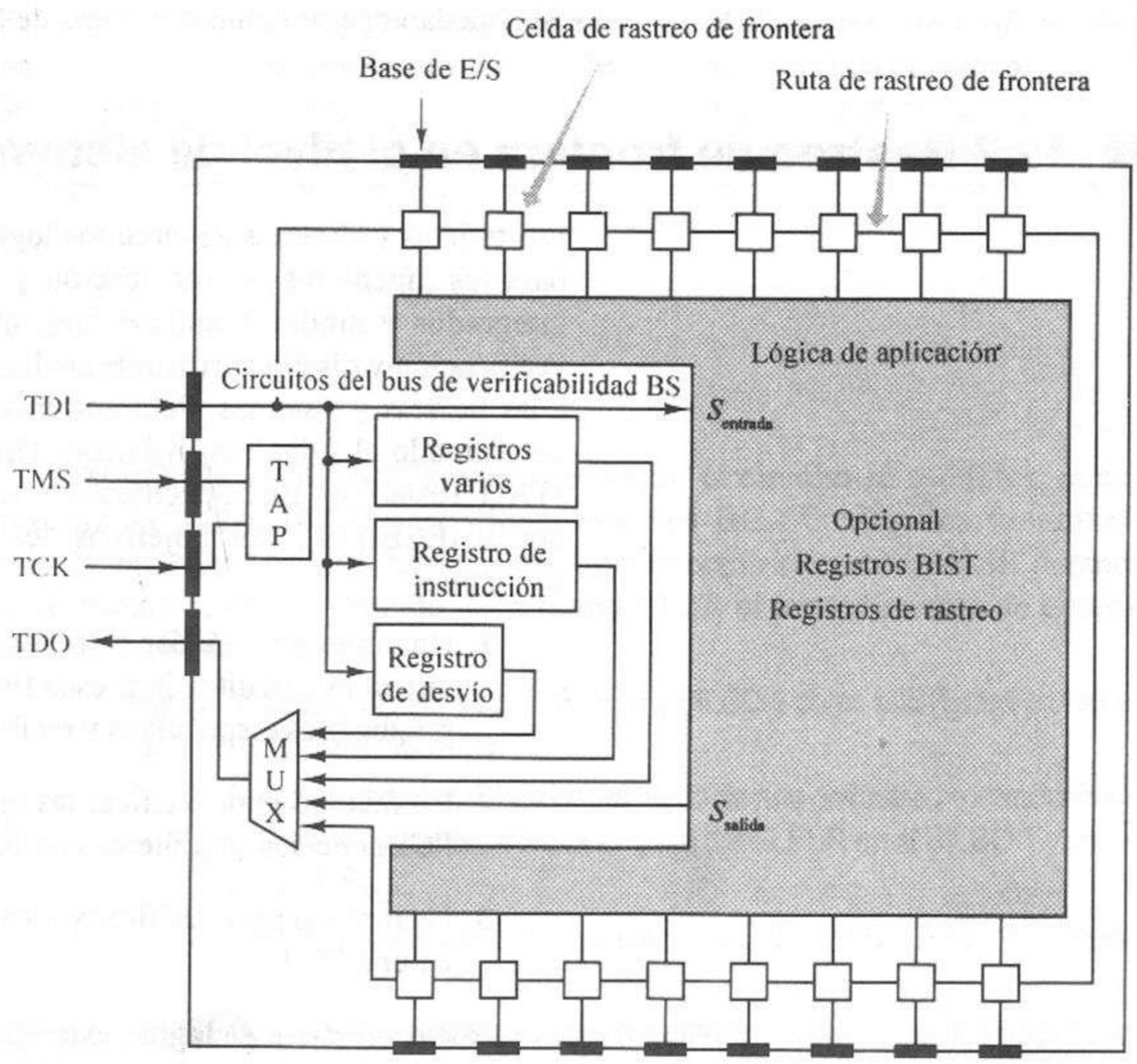


Figura 12.28 Circuito que se utiliza con el *bus* de verificabilidad 1149.1.

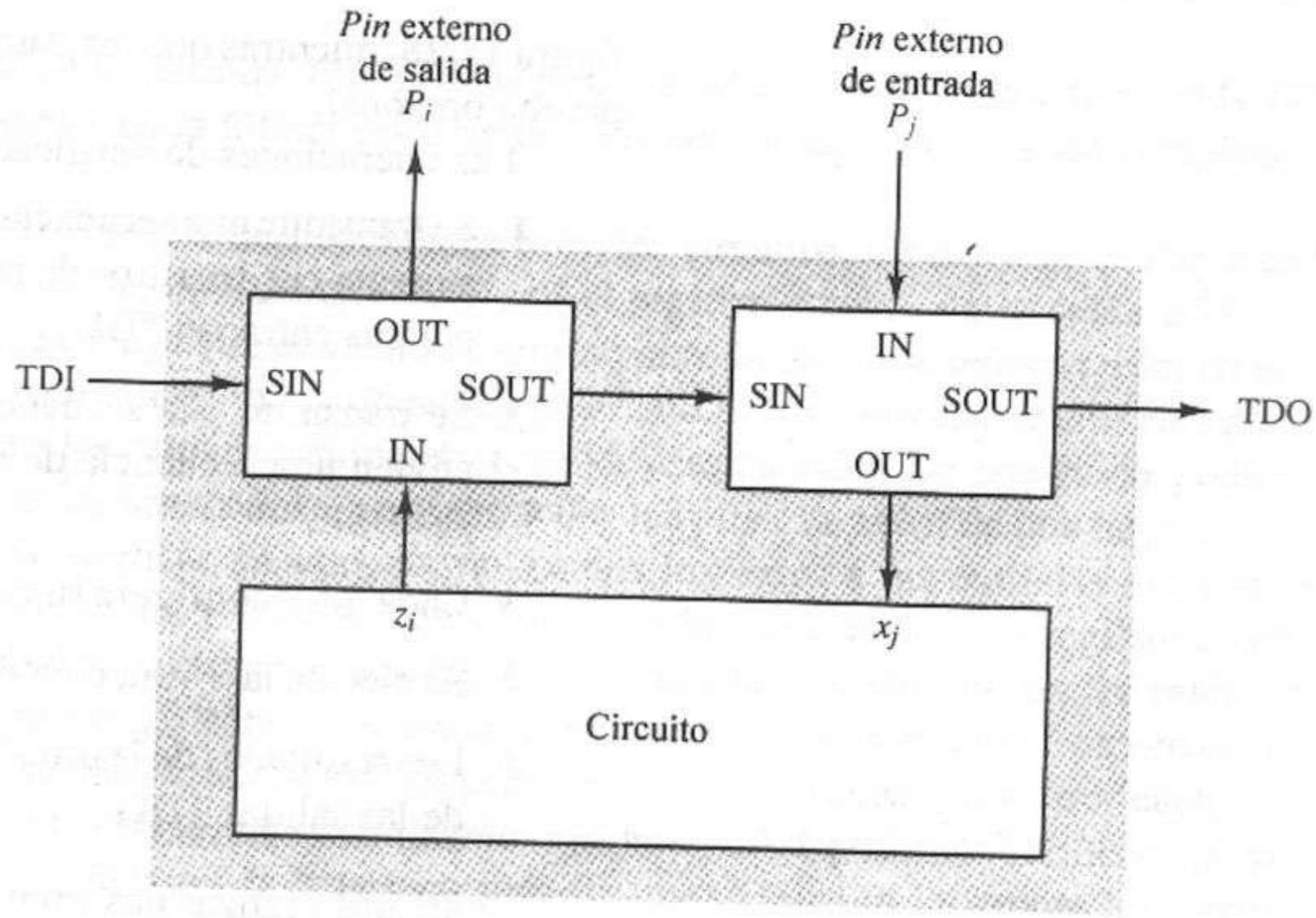


Figura 12.29 Celdas de rastreo de frontera entre las líneas de E/S del circuito y los *pines* externos.

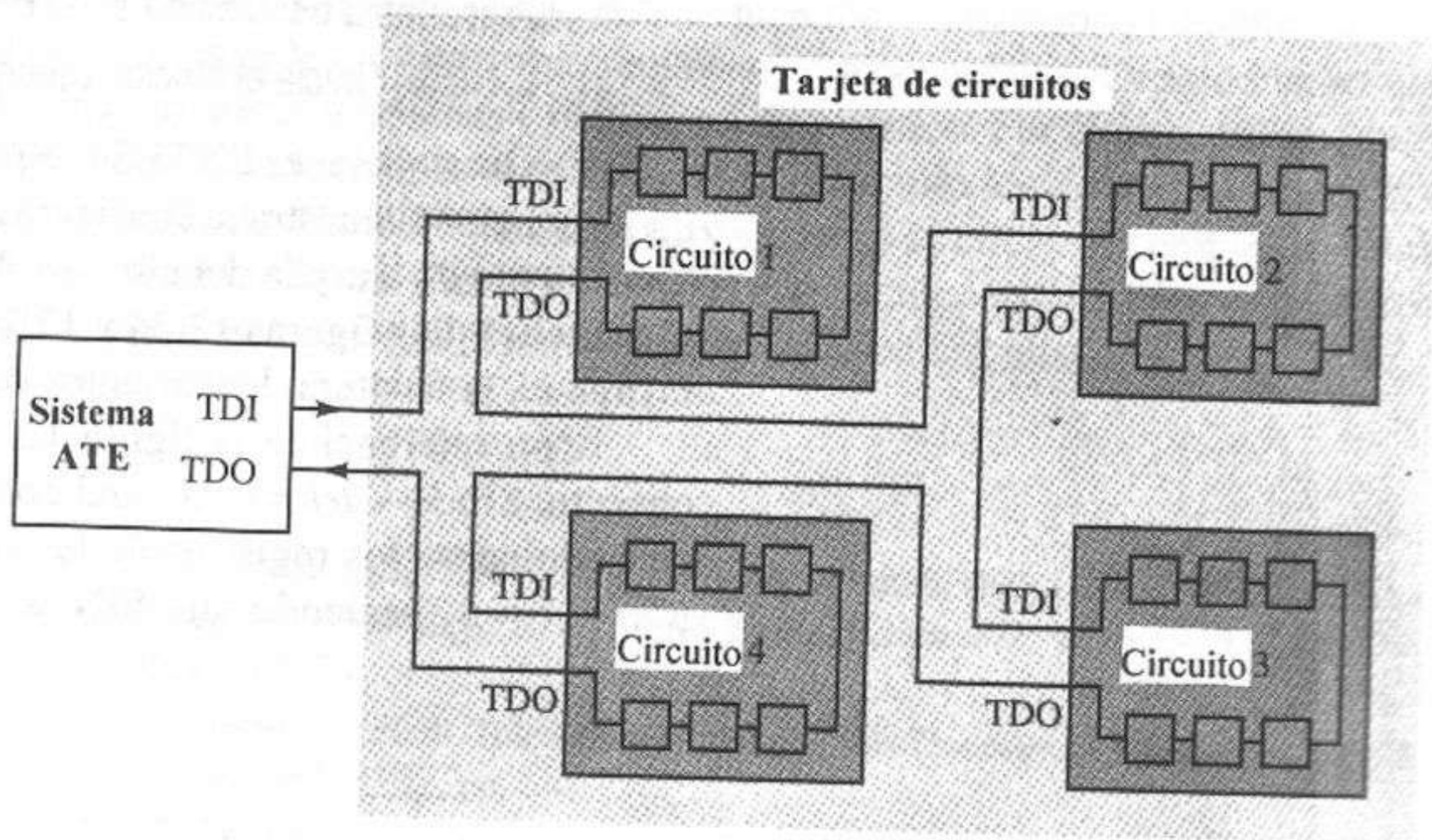


Figura 12.30 Tarjeta de circuitos configurada como un único registro de rastreo.

directamente al *pin* TDO de ese circuito, pasando sobre el BSR del circuito. De esta manera, el sistema ATE puede trabajar con ciertos circuitos durante la verificación.

El TAP tiene dos señales de entrada: TMS y TCK. TMS sirve para transmitir instrucciones del sistema ATE a los circuitos, sincronizadas mediante una señal de reloj en el *pin* TCK. Estas instrucciones hacen que los circuitos interpreten los bits recibidos en sus *pines* TDI como instrucciones o datos de prueba. Las instrucciones se envían a un registro especial de instrucciones, que aparece en la

figura 12.28, mientras que los datos se envían al BSR o a un registro de datos de prueba opcional.

Las operaciones de verificación se realizan como sigue.

1. Se transmite una secuencia de bits por TMS para obligar a los dispositivos a cargar sus registros de instrucción con una secuencia de bits enviados por sus entradas TDI.
2. Se transmite una secuencia de bits en TMS para que los dispositivos envíen una secuencia de datos de prueba desde sus entradas TDI hasta sus BSR.
3. Cada TAP configura su circuito según la instrucción de prueba recibida.
4. Se ejecuta la instrucción de prueba, capturando los resultados en el BSR.
5. Los resultados de la prueba se desplazan hacia el sistema ATE por medio de las salidas TDO.

Podemos realizar dos tipos generales de operaciones de verificación: las pruebas de interconexión de los circuitos y las pruebas internas de los circuitos. Las principales instrucciones de verificación son:

BYPASS: Pasa sobre el BSR enviando el TDI en forma directa al TDO.

EXTEST: Envía los datos del BSR a los *pines* de salida.

SAMPLE: Captura los datos de los *pines* de entrada en el BSR.

INTEST: Aplica los vectores de prueba al circuito interno desde el BSR y captura la respuesta del circuito en el BSR.

RUNBIST: Inicia el funcionamiento de los circuitos BIST en el circuito integrado.

Las instrucciones EXTEST, SAMPLE e INTEST implican el uso del BSR. La figura 12.31 muestra la configuración de una celda con rastreo en la frontera. En cada *pin* de entrada del circuito, IN es la entrada externa y OUT controla el *pin* externo. Las figuras 12.28 y 12.29 ilustran las relaciones entre las celdas de rastreo en la frontera, los circuitos de aplicación y los *pines* externos de E/S.

Con referencia a la figura 12.31, durante el funcionamiento normal del circuito, *Mode–Ctrl* = 0, lo cual conecta IN con OUT en cada celda del BSR, pasando sobre los registros de las celdas. Durante las operaciones de rastreo, *ShiftDR* = 1, haciendo que SIN se recorra a SOUT mediante el *flip–flop* de

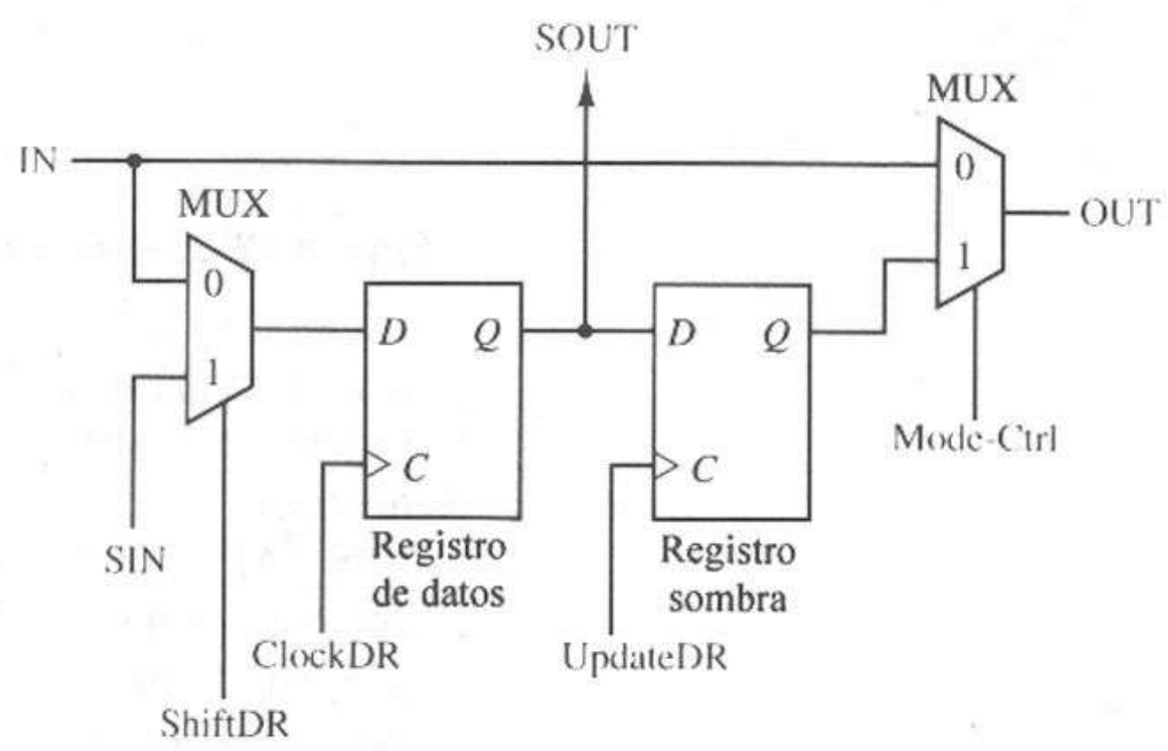


Figura 12.31 Celda de rastreo de frontera.

rastreo de cada celda. El SIN de la primera celda se conecta con la entrada TDI, y el SOUT de la última celda se conecta con la salida TDO para completar el BSR.

Al verificar las conexiones entre los circuitos, los datos se envían a cada *pin* de salida, conectando OUT con el registro sombra de la celda BSR. Los datos de cada *pin* de entrada se capturan introduciendo la señal de reloj IN en el registro de rastreo. A continuación, se examinan los datos capturados para garantizar que los datos transmitidos y recibidos sean iguales. De este modo, podemos verificar las interconexiones entre todos los *pines* de señal de una tarjeta.

Al verificar un circuito, aplicamos los datos a las entradas del circuito desde las celdas del BSR, conectando los bits adecuados del registro sombra con OUT, y capturamos los datos de las salidas del circuito en las celdas del BSR correspondiente conectando IN con el registro de rastreo. Los vectores de prueba se introducen en el BSR antes de aplicar la prueba, y los datos capturados se extraen del BSR para examinar los resultados. Así, podemos verificar cualquier circuito individual de la tarjeta, siempre que el circuito incorpore los circuitos de bus de verificabilidad 1149.1.

12.8 Resumen

En este capítulo presentamos los temas de diagnóstico de fallas de los circuitos lógicos y el diseño para la verificabilidad. En primer lugar presentamos un análisis general del diagnóstico de fallas. A continuación analizamos el diagnóstico de fallas para las redes lógicas combinatorias y secuenciales. Consideramos los métodos para generar pruebas y realizar un diagnóstico. Por último, presentamos los métodos que facilitan la verificación de un circuito lógico digital, tanto para circuitos individuales como para tarjetas y sistemas de circuitos completos. Estos métodos facilitan la verificación mediante un equipo automático y permiten realizar verificaciones completamente dentro de un circuito.

REFERENCIAS BIBLIOGRÁFICAS

1. B. D. ARMSTRONG, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets," *IEEE Trans. Computers*, Vol. EC-15, No.1, Feb. 1966, pp. 66–73.
2. A. AVIZIENIS, "Fault Tolerant Computing: An Overview," *Computer*, Vol. 5, No.1, Jan.–Feb. 1971, pp. 5–7.
3. D. C. BOSSEN AND S. J. HONG, "Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks," *IEEE Trans. Computers*, Vol. C-20, No 11, Nov. 1971, pp. 1252–1257.
4. W. G. BOURICIUS, E. P. HSIEH, G. R. PUTZOLU, J. P. ROTH, P. R. SCHNIEDER, AN C. J. TAN, "Algorithms for Detection of Faults in Logic Circuits," *IEEE Tra Computers*, Vol. C-20, No. 11, Nov. 1971, pp. 1258–1264.
5. M. A. BREUER, "A Random and an Algorithmic Technique for Fault Detec Test Generation for Sequential Circuits," *IEEE Trans Computers*, Vol. C-20 11, Nov. 1971, pp. 1364–1371.
6. H. Y. CHANG, E. G. MANNING, AND G. METZE, *Fault Diagnosis of D Systems*. New York: Wiley (Interscience Division), 1970.
7. A. D. FRIEDMAN AND P. R. MENON, *Fault Detection in Digital Circuits* wood Cliffs, NJ: Prentice Hall, 1971.

8. F. C. HENNIE, *Finite-state Models for Logical Machines*. New York: Wiley, 1968.

9. W. H. KAUTZ, "Fault Testing and Diagnosis in Combinational Digital Circuits," *IEEE Trans. Computers*, Vol. C-17, No. 4, Apr. 1968, pp. 352–367.

10. Z. KOHAVI AND P. LAVALLEE, "Design of Sequential Machines with Fault Detection Capabilities," *IEEE Trans. Computers*, Vol. EC-16, No. 4, Aug. 1967, pp. 473–484.

11. E. F. MOORE, "Gedanken-Experiments on Sequential Machines," in *Automata Studies*, C. E. Shannon, ed. Princeton, NJ: Princeton University Press, 1956.

12. J. P. ROTH, W. G. BOURICIUS, AND P. R. SCHNEIDER, "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," *IEEE Trans. Computers*, Vol. EC-16, No. 5, Oct. 1967, pp. 567–580.

13. W. W. PETERSON AND E. J. WELDON, JR., *Error-correcting Codes*, 2nd ed. Cambridge, MA: MIT Press, 1972.

14. B. KOENEMANN, J. MUCHA, AND G. ZWIEHOFF, "Built-in Logic Block Observation Techniques," *Digest of Papers, 1979 Test Conference*, IEEE Pub. 79CH1509-9C, Oct. 1979, pp. 37–41.

15. L. H. GOLDSTEIN, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. Computers*, Vol. CAS-26, No. 9, Sept. 1979, pp. 685–693.

16. A. MICZO, *Digital Logic Testing and Simulation*. New York: Harper & Row, 1986.

17. M. ABRAMOVICI, M. BREUER, AND A. D, FRIEDMAN, *Digital Systems Testing and Testable Design*, New York: Computer Science Press, 1990.

18. *TEST ACCESS PORT AND BOUNDARY-SCAN ARCHITECTURE*, IEEE Standard 1149.1-1990, IEEE Standards Board, 345 East 47th St., New York, NY 10017, May 1990.

19. K. P. PARKER, *The Boundary-scan Handbook*. Norwell, MA: Kluwer Academic Pub., 1992.

20. J. E. SMITH, "Measures of Effectiveness of Fault Signature Analysis," *IEEE Trans. Computers*, Vol. C-29, No. 6, June 1980, pp. 510–514.

PROBLEMAS

12.1 Determine la función realizada por la red de la figura P12.1 para cada una de las fallas siguientes:

(a) 1/0. **(c)** 3/0.

(b) 1/1. **(d)** 4/1.

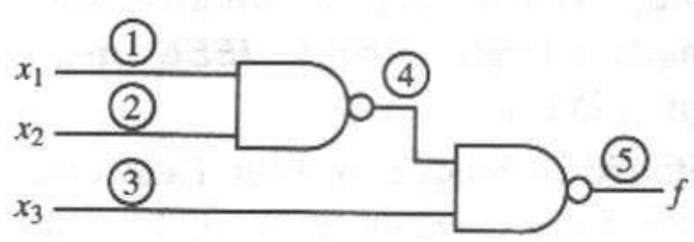


Figura P12.1

12.2 Determine la función realizada por la red de la figura P12.2 para cada una de las fallas siguientes:

(a) 2/0. **(c)** 9/0.

(b) 8/1. **(d)** 5/0.

12.3 Utilice el método del OR exclusivo para determinar todas las pruebas para las fallas del problema 12.1.

12.4 Repita el problema 12.3 usando el método de sensibilización de trayectorias.

12.5 Utilice el método del OR exclusivo para determinar todas las pruebas para las fallas enumeradas en el problema 12.2.

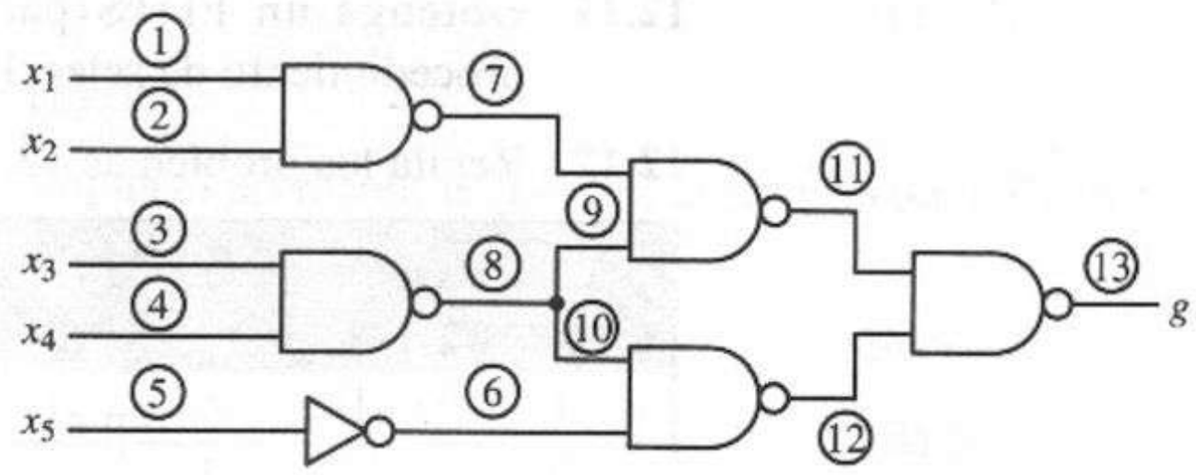


Figura P12.2

12.6 Repita el problema 12.5 usando el método de sensibilización de trayectorias.

12.7 Enumere todas las fallas que pueden detectar las siguientes pruebas para la red de la figura P12.2.

(a) 10101.

(b) 10100.

(c) 11011.

(d) 11010.

12.8 Construya una tabla de fallas que contenga todas las fallas posibles y todas las combinaciones de entradas posibles para la red de la figura P12.1.

12.9 Construya una tabla de fallas que sólo contenga las fallas de los puntos de verificación, para la red de la figura P12.9.

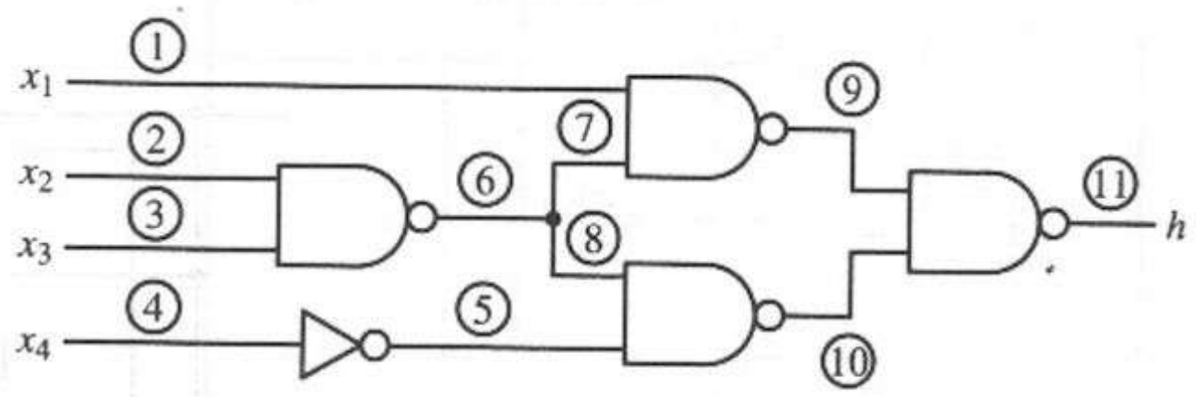


Figura P12.9

12.10 Determine un FDTS mínimo para la siguiente tabla de fallas:

Pruebas			Fallas							
x_1	x_2	x_3	a	b	c	d	e	f	g	h
0	0	0	1							1
0	0	1		1				1		
0	1	0	1		1				1	
0	1	1				1		1		
1	0	0	1							1
1	0	1		1			1		1	
1	1	0						1		1
1	1	1	1			1			1	1

12.11 Obtenga un FDTS para la tabla del problema 12.10 usando el procedimiento de selección cercano al mínimo.

12.12 Repita los problemas 12.10 y 12.11 para la siguiente tabla de fallas:

	Fallas								
Pruebas	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
1	1		1						
2		1		1		1			
3			1		1				
4				1					1
5					1		1		
6		1							1
7							1	1	
8	1							1	

12.13 Determine todas las pruebas que pueden servir para distinguir cada uno de los siguientes pares de fallas en la red de la figura P12.9.

(a) 1/0–4/1.
(b) 2/0–3/0.
(c) 7/0–8/0.
(d) 1/1–7/1.

12.14 Dado el circuito de la figura P12.14, determine una secuencia de inicialización y una secuencia de observación para cada una de las fallas siguientes:

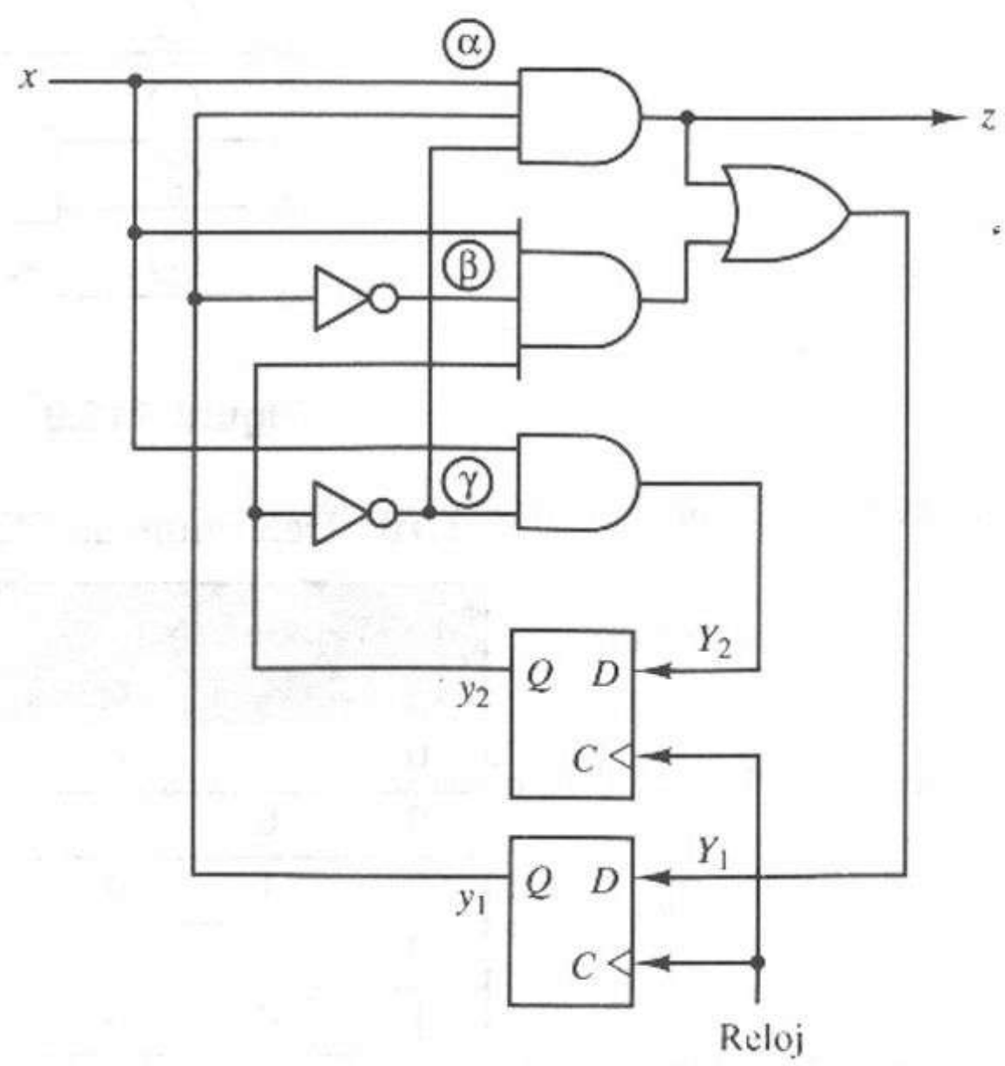


Figura P12.14

(a) $\alpha/1$.

(b) $\beta/0$.

(c) $\gamma/1$.

12.15 Repita el problema 12.14 para el circuito de la figura P12.15 y las fallas siguientes:

(a) $\alpha/1$.

(b) $\beta/0$.

(c) $\gamma/0$.

(d) $\delta/1$.

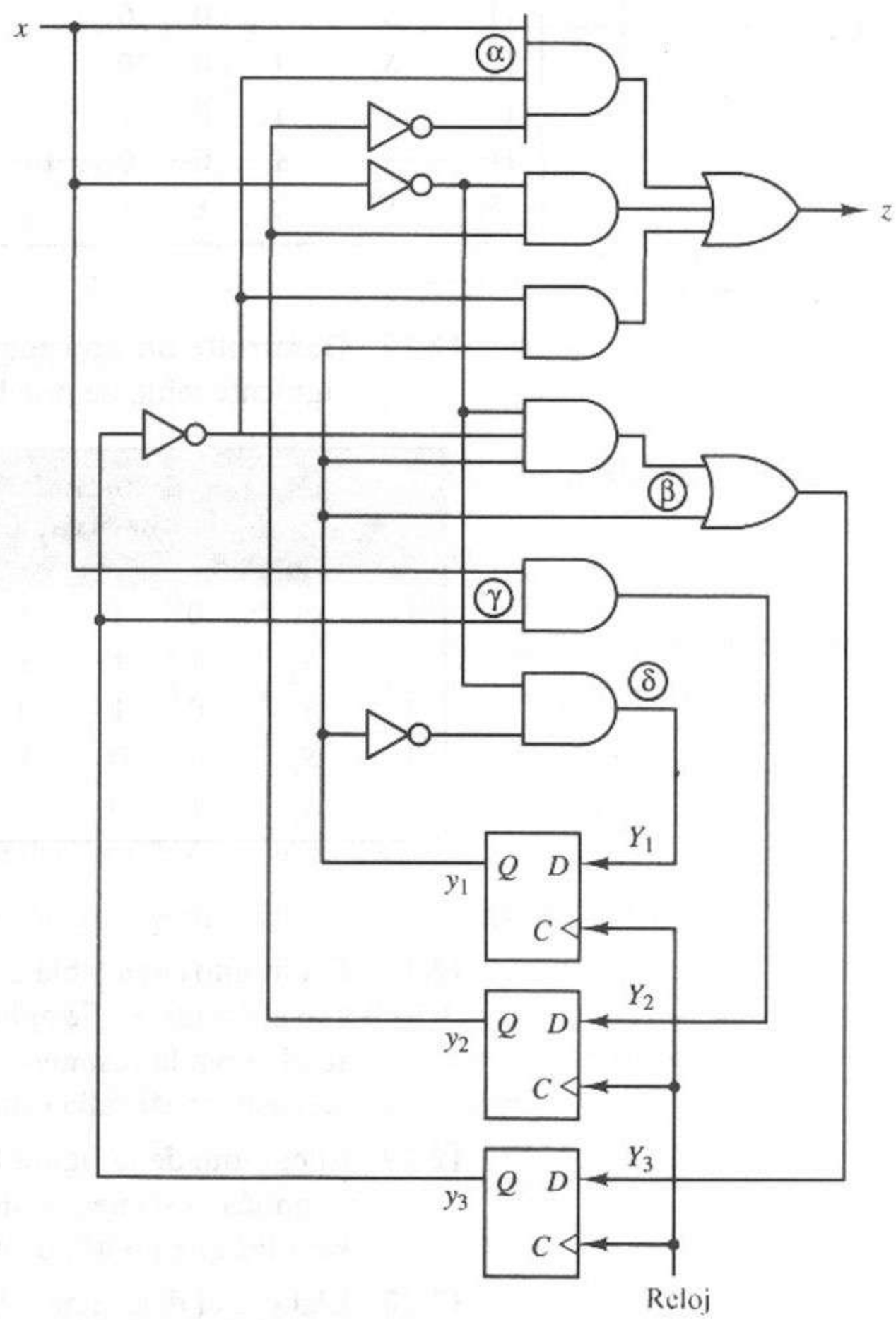


Figura P12.15

12.16 Reduzca la siguiente tabla de fallas hipotética, aprovechando la cobertura de pruebas.

	Estado inicial	Secuencia de observación, x				Secuencia de salida, z				Estado final	Fallas verificadas
1	S_1	1				$\bar{D}$				S_2	a_0, c_1
2	S_2	0				$\bar{D}$				S_2	b_0, i_0
3	S_3	1				$\bar{D}$				S_3	g_0, f_0
4	S_4	0				D				S_1	h_1, b_1
5	S_2	1				$\bar{D}$				S_3	l_0, e_0
6	S_1	1	1			0	$\bar{D}$			S_3	j_0, g_1
7	S_3	1	0			0	$\bar{D}$			S_4	n_0, j_1
8	S_3	0	0			0	D			S_1	p_0, m_0
9	S_2	1	0			0	$\bar{D}$			S_4	f_1, h_0
10	S_4	0	1	1		1	0	$\bar{D}$		S_3	a_1, i_1
11	S_2	1	0	0		0	0	D		S_1	n_1, k_0
12	S_3	1	0	0		0	0	D		S_1	p_1, l_1
13	S_1	1	0	1		0	0	$\bar{D}$		S_3	d_0, k_1
14	S_2	1	0	0	1	0	0	1	$\bar{D}$	S_2	m_1, d_1
15	S_4	0	1	1	1	1	0	0	$\bar{D}$	S_3	e_1, c_0

12.17 Desarrolle un procedimiento de verificación para un circuito con la siguiente tabla de pruebas reducida:

	Estado inicial	Secuencia de observación, x				Secuencia de salida, z			Estado final	Fallas verificadas
1	S_1	0	1	1	0	$\bar{D}$	$\bar{D}$		S_3	a_0, d_1, e_1, h_0
2	S_2	1	1	1	$\bar{D}$	1	D		S_1	b_1, e_0
3	S_3	0	1	0	0	1	$\bar{D}$		S_4	c_0, f_0
4	S_4	0	0	1	0	$\bar{D}$	D		S_1	a_1, c_1, f_1, g_0, h_1
5	S_1	1	0		0	$\bar{D}$			S_2	b_0, d_0, g_1

12.18 El circuito cuya tabla de pruebas se da en el problema 12.17 se inicializa con el estado S_1. Se aplica una secuencia de entrada 011010 al circuito y se observa la respuesta de salida 001010. ¿Tiene fallas el circuito? En tal caso, ¿cuál falla está presente?

12.19 El circuito de la figura P12.19 se inicializa con el estado $y_1 = 0, y_2 = 1$ y se aplica la secuencia de entrada $x = 010111$. Determine todas las fallas simples que podría detectar esta secuencia.

12.20 Elabore el diagrama y la tabla de estados para la figura 12.24a.

12.21 Para el ALFSR de la figura P12.21:

(a) Construya el diagrama y la tabla de estados.

(b) Calcule la probabilidad de omisión de errores para el circuito.

(c) Determine las respuestas correcta e incorrecta para las secuencias de entrada seriales de las figuras 12.24b y c.

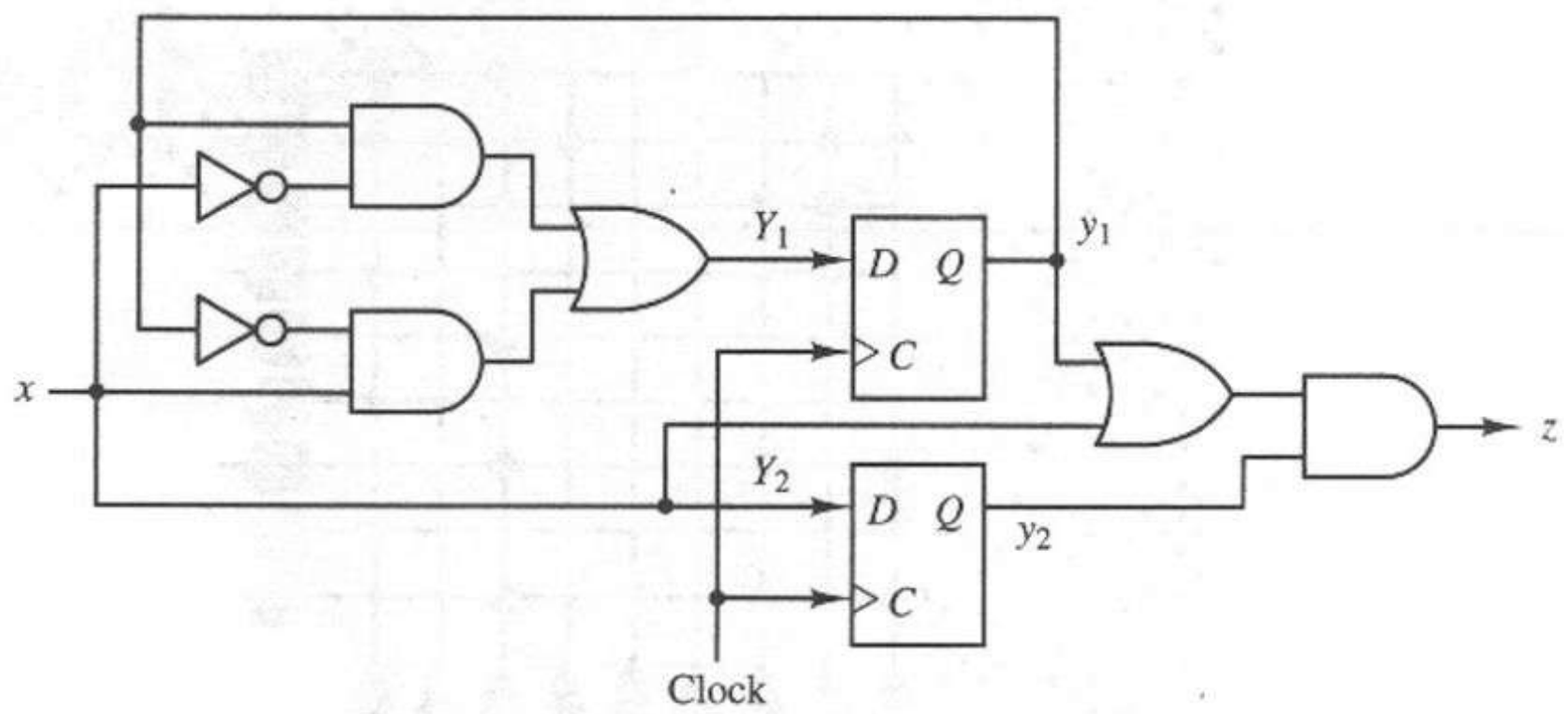


Figura P12.19

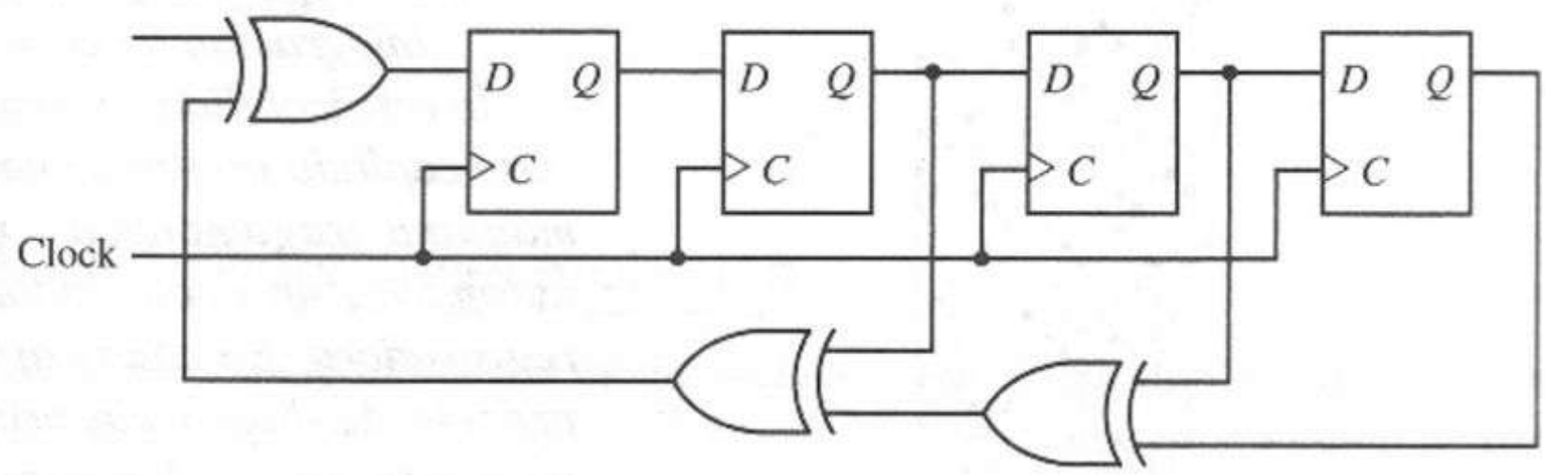


Figura P12.21

12.22 Diseñe un registro BILBO de 8 bits con la siguiente tabla funcional:

$B1$	$B2$	Función
0	0	Modo AFLSR
0	1	Modo de rastreo (registro de corrimiento)
1	0	Modo normal (carga paralela)
1	1	Modo MISR

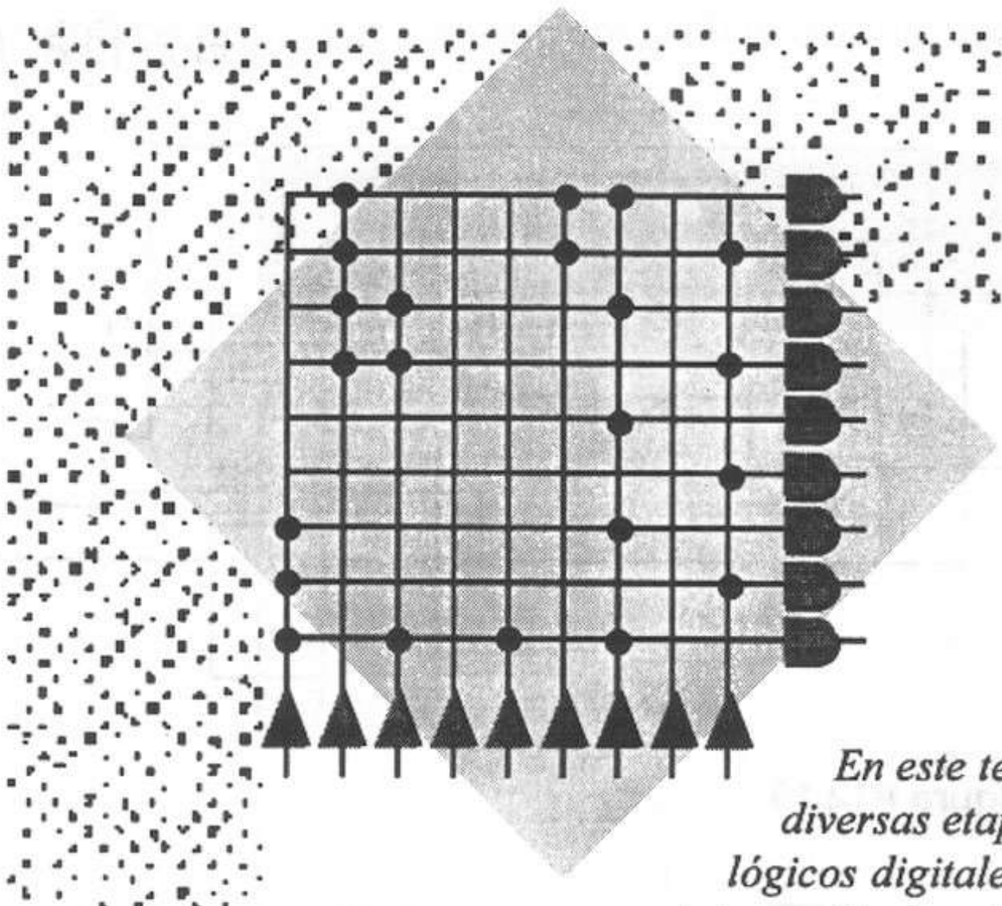
Sea $p(x) = 1 + x + x^5 + x^6 + x^8$.

12.23 Diseñe un registro BILBO de 10 bits con la siguiente tabla funcional:

$B1$	$B2$	$B3$	Función
0	0	0	Modo de rastreo (registro de corrimiento)
0	1	0	*Reset* síncrono
1	0	0	Modo MISR
1	0	1	Modo ALFSR
1	1	0	Modo normal (carga paralela)

Sea $p(x) = 1 + x^3 + x^{10}$.

13



En este texto hemos presentado métodos para las diversas etapas del análisis y diseño de los circuitos lógicos digitales. Los problemas "reales" requieren la integración de estos métodos, considerando el costo total, la verificabilidad y otros aspectos prácticos. Como ilustración, este capítulo presenta cuatro casos de diseño: un juego con una máquina tragamonedas, un sistema de acceso sin llave a un automóvil, un controlador de tráfico de un carril y una caja registradora. En cada caso analizaremos todos los aspectos del proceso de diseño. En primer lugar, definiremos el problema y determinaremos los requisitos del sistema. A continuación determinaremos los módulos funcionales necesarios para cumplir con los requisitos, diseñando un circuito lógico o eligiendo un componente estándar para implantar cada módulo. Por último, integraremos los módulos en un sistema completo. Aunque en estos ejemplos utilizaremos los módulos estándar TTL, el lector puede considerar también la lógica programable y otros métodos.

Ejemplos de diseño

13.1 Máquina tragamonedas electrónica

13.1.1 Definición del problema

Una *máquina tragamonedas* es un juego de tipo casino en el que un jugador deposita una moneda y jala una palanca, lo cual hace que un conjunto de ruedas comience a girar. Cada rueda tiene varios dígitos o imágenes pintadas en su circunferencia, que se despliegan a través de una ventana. Después de soltar la palanca, las ruedas se detienen y cada una muestra un dígito o imagen. Si los números o imágenes mostrados concuerdan con cierto conjunto de patrones específicos, se gana una cantidad determinada de dinero. La *recompensa*, o la cantidad de dinero ganada, depende del patrón concordante y, en algunos casos, puede depender de la cantidad de dinero depositada (o apostada).

Existen muchas variantes de este juego, como los juegos electrónicos de mano que no utilizan dinero, los cuales operan al jalar una palanca u oprimir un botón para iniciar el giro y detener las ruedas numéricas. A fin de minimizar las partes mecánicas, con frecuencia se utiliza un *display* electrónico para simular las ruedas que giran, cambiando de manera continua los dígitos desplegados. Las combinaciones ganadoras se indican mediante luces parpadeantes o alarmas audibles. En algunos casos, se ganan y pierden puntos, lo que permite a los jugadores mantener una puntuación mientras juegan.

En este ejemplo diseñaremos un juego de máquina tragamonedas electrónico operado al oprimir un interruptor JUGAR, lo que hará que los dígitos del intervalo [1..7] se muestren y cambien de manera continua en un panel electrónico a modo de dar la impresión de ruedas giratorias. Los dígitos se congelarán en el *display* al soltar el botón JUGAR. Si aparece alguno de los números de cierto conjunto, se otorgará una *recompensa*, un número dado de puntos. En nuestra máquina, cualquier combinación de dos o tres dígitos idénticos será *ganadora*, y tres dígitos idénticos devolverán una recompensa mayor (más puntos) que dos dígitos idénticos. El monto de la recompensa también dependerá del valor del dígito concordante y de una apuesta, que se fijará mediante dos interruptores antes de oprimir el botón JUGAR. La figura 13.1 muestra la distribución básica del panel frontal del juego.

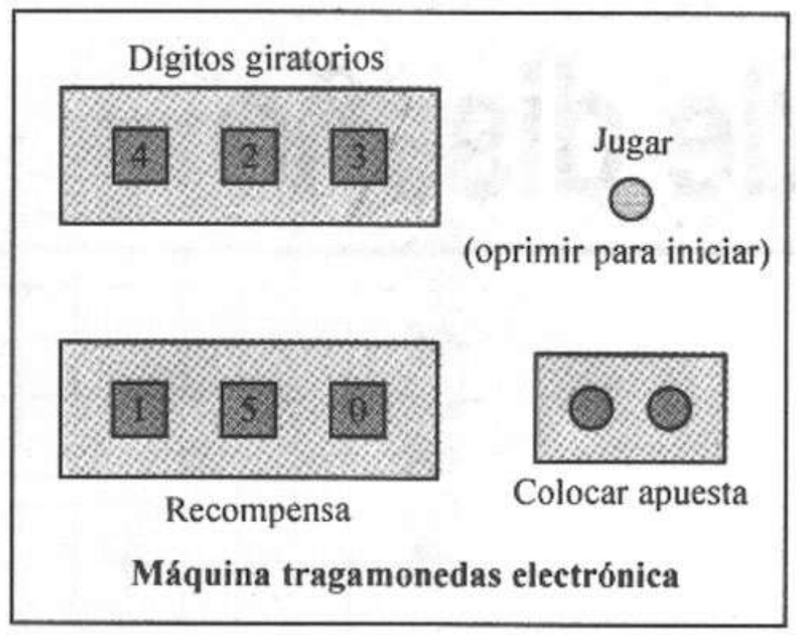


Figura 13.1 Juego electrónico de máquina tragamonedas.

13.1.2 Necesidades del sistema y plan de solución

Comenzaremos el proyecto determinando las necesidades del sistema y desarrollando un plan que satisfaga estas necesidades. Según la descripción del juego, necesitaremos varios subsistemas principales: dos *displays* electrónicos de tres dígitos para los dígitos giratorios y la recompensa, un botón JUGAR, dos interruptores para fijar la apuesta, circuitos para generar los números del *display* giratorio, y un circuito para identificar los patrones ganadores y calcular la recompensa. El diagrama de bloques de la figura 13.2 ilustra estos subsistemas, que describimos a continuación.

1. **Botón JUGAR**. Utilizaremos un interruptor de botón para el botón JUGAR de la máquina tragamonedas. El botón se oprime para restablecer el juego e iniciar los generadores de números aleatorios; el botón se suelta para detenerlos, y en ese momento se tomará una decisión en el sentido de que el jugador ha perdido o ganado.
2. ***Display* con dígitos giratorios**. Utilizaremos tres LED (diodos emisores de luz) de siete segmentos para mostrar los dígitos producidos por los generadores de números. Éstos se actualizarán de manera continua mientras el botón JUGAR se mantenga oprimido a fin de simular las ruedas de números giratorias. Como apoyo a este efecto, los tres dígitos cambiarán con diferentes frecuencias y también se detendrán en instantes diferentes después de que el botón JUGAR se haya soltado, para simular el hecho de que las ruedas se detienen una tras otra.
3. **Generadores de números**. Tres generadores de números forman la base del juego; cada uno produce dígitos en el intervalo [1..7]. Los generadores de números comienzan a funcionar cuando se oprime el botón JUGAR y se detienen después de que el botón JUGAR se libera. Para que los generadores de números parezcan aleatorios, dificultando con ello la predicción de sus valores finales, recibirán pulsos de reloj a una velocidad alta, y cada uno cambiará con una velocidad diferente. Además, al detenerse cada generador de números en un instante diferente después de soltar el botón JUGAR, será más difícil predecir los valores finales de los dígitos.
4. **Módulo temporizador**. El módulo temporizador contendrá un circuito oscilador y varios circuitos temporizadores para producir señales que controlen a los generadores de números. Un conjunto de salidas del circuito temporizador iniciará el funcionamiento de los generadores de números cuando se oprima el botón JUGAR y los detendrá en instantes

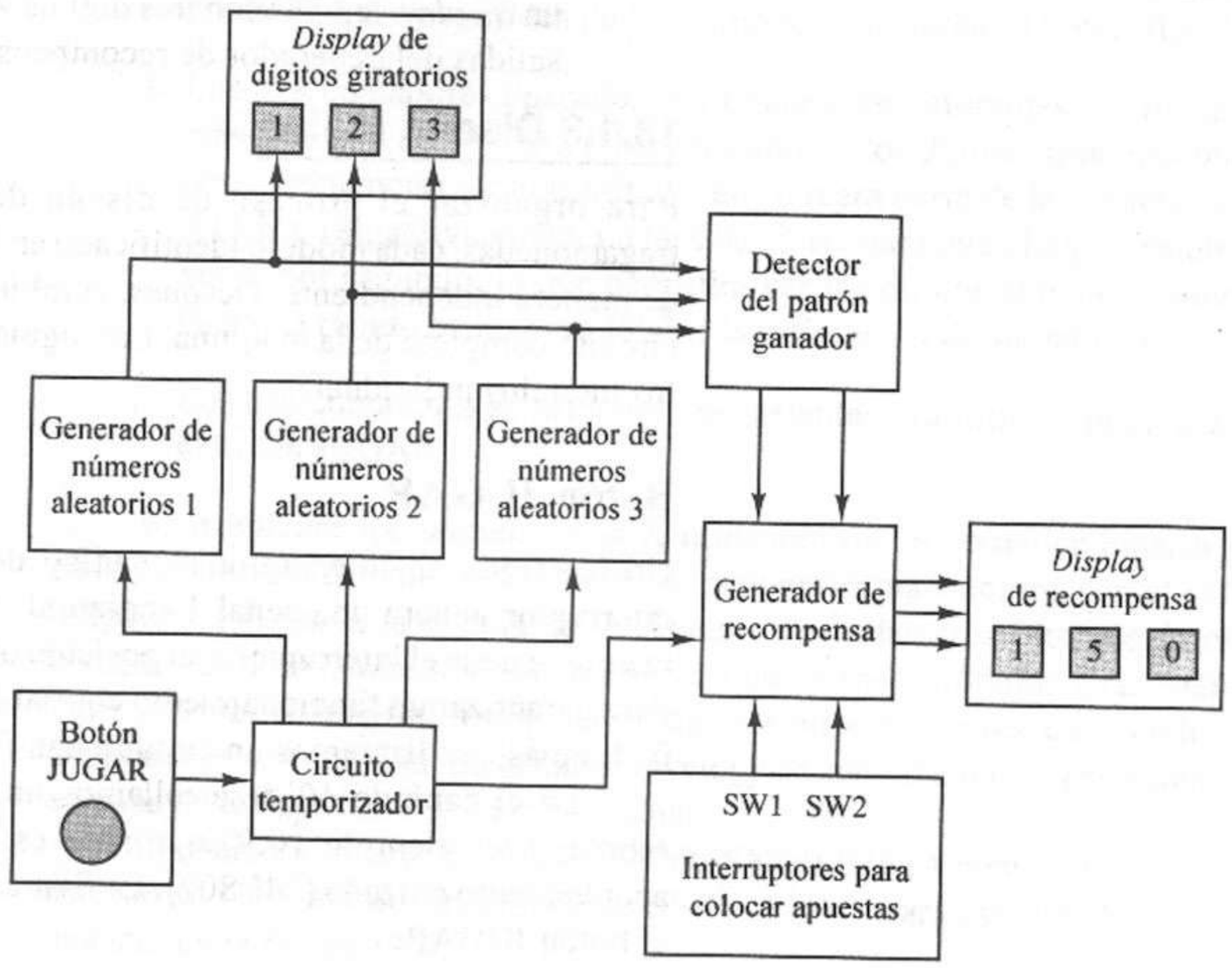


Figura 13.2 Diagrama de bloques para el juego electrónico de máquina tragamonedas.

diferentes cuando se suelte el botón JUGAR. Un segundo conjunto de salidas hará que los generadores de números cambien con velocidades distintas.

5. **Entradas para la apuesta**. El número de puntos obtenidos se multiplicará por un factor de escala en el intervalo [1..4] seleccionado mediante dos interruptores que se establecerán antes de oprimir el botón JUGAR. Esto simula la colocación de una apuesta respecto del resultado del juego. Podemos suponer que se pierde cierta cantidad de puntos si no ocurre una combinación ganadora, de modo que la pérdida dependa también de la apuesta.
6. **Detector de un patrón ganador**. Este módulo evaluará las salidas de los tres generadores de números, después de que todos ellos se hayan detenido, si los tres números finales producidos por los generadores de números corresponden a una de las combinaciones ganadoras; es decir, para determinar si dos o tres números concuerdan. Los resultados se enviarán al generador de la recompensa.
7. **Generador de recompensa**. El generador de recompensa calculará la recompensa (el número de puntos obtenidos) con base en la detección de una combinación ganadora y el valor del dígito concordante. La recompensa será proporcional al número del dígito concordante. Además, la recompensa por tres dígitos concordantes será mayor que la correspondiente a dos dígitos concordantes. Por último, la recompensa se multiplicará por un factor de escala (1 a 4) correspondiente a la cantidad apostada. Se mostrará una recompensa nula (000) si no se detecta un patrón ganador.

8. ***Display* de recompensa**. El número de puntos obtenido se exhibirá en un *display* de LED con tres dígitos y siete segmentos, controlado por las salidas del generador de recompensa.

13.1.3 Diseño lógico

Para organizar el proceso de diseño del circuito lógico de la máquina tragamonedas, cada módulo identificado en las necesidades se diseñará y probará de manera independiente. Después, combinaremos estos módulos para crear el circuito completo de la máquina. Los siguientes párrafos presentan el diseño de los módulos individuales.

Botón JUGAR

Utilizaremos un interruptor mecánico de botón para el botón JUGAR. El interruptor genera una señal 1 lógico al oprimirse y 0 lógico al soltarse. Un resorte regresa el interruptor a su posición original si no se le mantiene oprimido. Para garantizar un funcionamiento confiable, es decir, transiciones $0 \rightarrow 1$ y $1 \rightarrow 0$ "limpias", utilizaremos un circuito con filtrado de rebotes.

En el capítulo 10 desarrollamos un circuito interruptor con filtrado de rebotes (ver ejemplo 10.9) que sólo es un par de compuertas NAND con acoplamiento cruzado (74LS00). La figura 13.3 ilustra el circuito completo para el botón JUGAR.

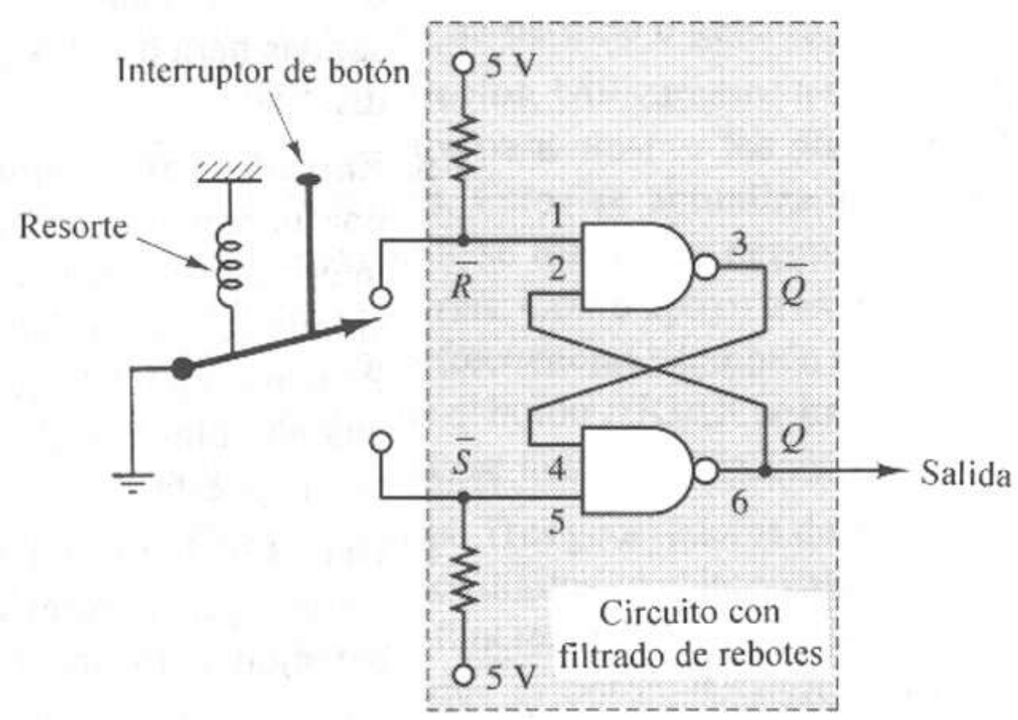


Figura 13.3 Botón JUGAR y circuito con filtrado de rebotes.

Generadores de números

Un verdadero generador de números aleatorios sería ideal para este juego, pero no es un circuito trivial. En vez de él, utilizaremos un circuito contador binario síncrono simple que cuente de manera continua de 1 a 7 para cada uno de los tres dígitos. La apariencia aleatoria provendrá del funcionamiento de un circuito temporizador que hará tres cosas:

1. Los contadores trabajarán con tal rapidez que dificultarán al jugador la predicción de las salidas de los contadores al soltar el botón JUGAR.

2. Los tres contadores operarán con frecuencias diferentes. La frecuencia más alta será de 45 hertz (Hz) para uno de los dígitos, mientras que las otras frecuencias serán la mitad y una tercera parte de la primera, o 22.5 y 15 Hz, respectivamente. La tasa de 15 Hz hará que el dígito cambie 15 veces por segundo, lo que permitirá ver los dígitos aunque cambien lo bastante rápido como para inhibir la estimación de su valor final.

3. Los tres contadores se detendrán en instantes distintos después de soltar el botón JUGAR.

Para obtener los números 1 a 7, utilizaremos un contador binario para cada generador de números, que será cargado en forma síncrona con el valor 1 después de llegar a la cuenta de 7. Aunque bastarían tres bits, elegimos un contador de 4 bits, pues los contadores de 3 bits no son módulos TTL estándar. Para obtener diferentes frecuencias de conteo e instantes de conclusión, necesitamos entradas independientes de reloj y de activación para cada contador, además de la capacidad de carga paralela.

El contador binario síncrono de 4 bits SN74LS163A cumple los requisitos anteriores. Utilizaremos tres módulos SN74LS163A para este diseño, como se muestra en la figura 13.4.

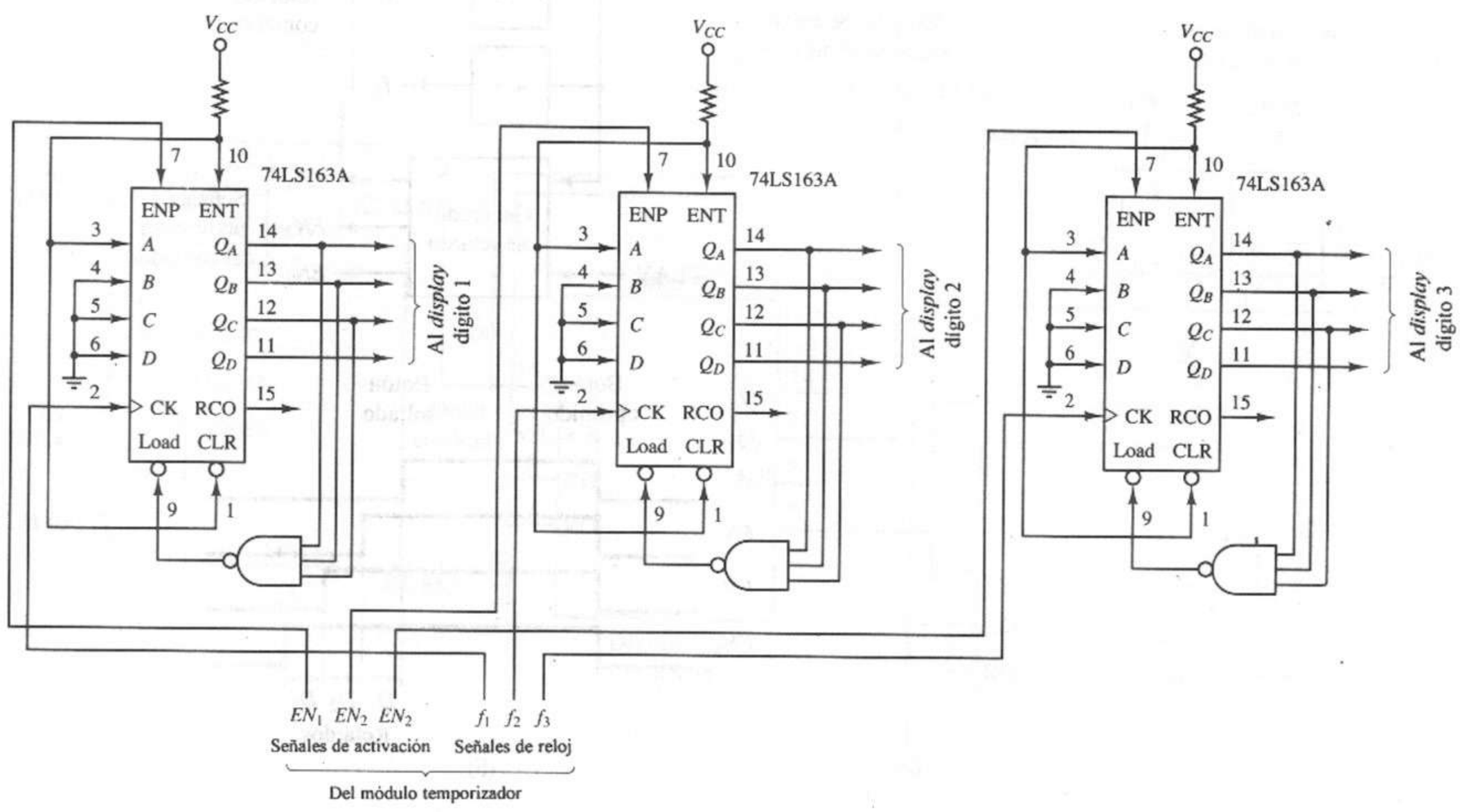


Figura 13.4 Generación de números: circuitos contadores 1 a 7.

Observe que necesitamos tres señales para activar el conteo $\{EN_1, EN_2, EN_3\}$ y tres diferentes señales de reloj $\{f_1, f_2, f_3\}$ para controlar los tres contadores. Las salidas del contador se envían al *display* giratorio digital y al detector de la combinación ganadora.

Módulo temporizador

La figura 13.5a muestra un diagrama de bloques del módulo temporizador. Por la figura 13.4, sabemos que necesitamos tres señales de reloj $(f_1, f_2, f_3\}$ y tres señales de activación (EN_1, EN_2, EN_3) para controlar los contadores del generador de números. Diseñaremos un circuito oscilador para obtener una onda cuadrada de frecuencia $f_1 = 45$ Hz que usaremos como señal de reloj para el primer contador. Dividiremos entonces la frecuencia de esta señal entre 2 y 3 a fin de producir otras dos señales con frecuencias $f_2 = 22.5$ Hz y $f_3 = 15$ Hz, respectivamente. El diagrama de tiempos de la figura 13.5 ilustra el funcionamiento deseado de las tres señales de activación. Las tres señales de activación asumirán el valor 1 al dejar de oprimir el botón JUGAR. Después de soltar el botón, EN_1 asumirá el

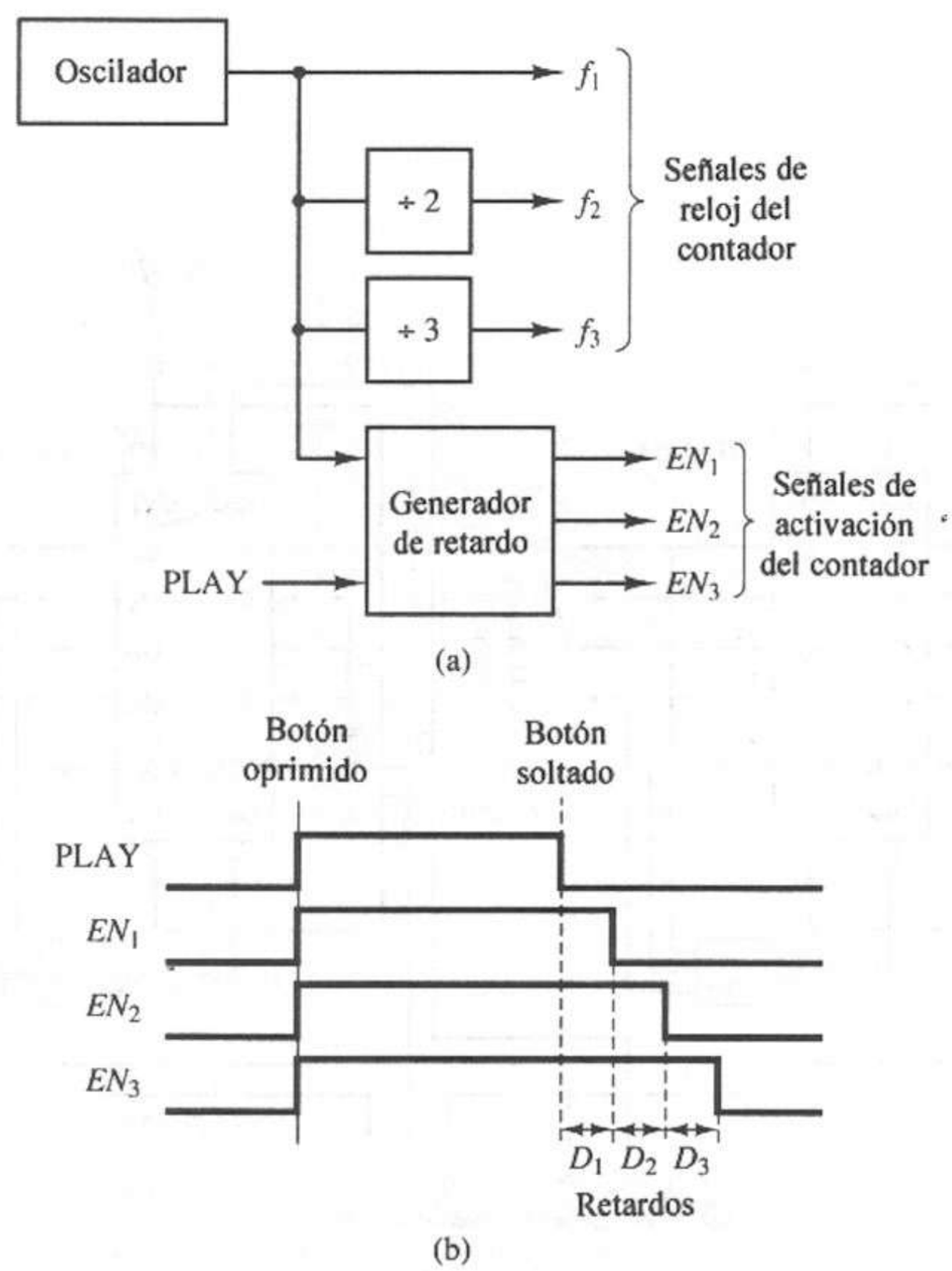


Figura 13.5 Módulo temporizador para el generador de números. **(a)** Generador de tiempos. **(b)** Tiempos para la señal de activación.

valor 0 después de un corto retardo D_1, EN_2 lo hará después de un segundo retardo D_2, y EN_3 después de un tercer retardo D_3.

Implantaremos el oscilador mediante un módulo temporizador de precisión 555 configurado para operar en modo *multivibrador astable*, como describimos en el capítulo 6 e ilustramos en la figura 13.6. La frecuencia de oscilación será de 45 Hz; en el capítulo 6 definimos esta frecuencia como

$$f = \frac{1.44}{(R_A + 2R_B)C} \tag{13.1}$$

Por tanto, para obtener $f = 45$ Hz, elegimos los siguientes valores para los componentes:

$$R_A = 15k\Omega$$
$$R_B = 8.5k\Omega$$
$$C = 1\,\mu F$$

Utilizaremos un circuito de división entre 2 y un circuito de división entre 3 para generar las otras dos señales de reloj. Como podemos dividir de manera conveniente las frecuencias de los trenes de pulsos mediante contadores binarios, utilizaremos un contador binario 74LS92 módulo 12 como muestra la figura 13.6. Las salidas del 74LS92, Q_A, Q_B, Q_C y Q_D, son simplemente trenes de pulsos cuyas frecuencias son 1/2, 1/3, 1/6 y 1/12 de la entrada de reloj, respectivamente. Por tanto, la salida Q_A proporcionará $f_2 = f_1/2 = 22.5$ Hz, mientras que Q_B proporcionará $f_3 = f_1/3 = 15$ Hz.

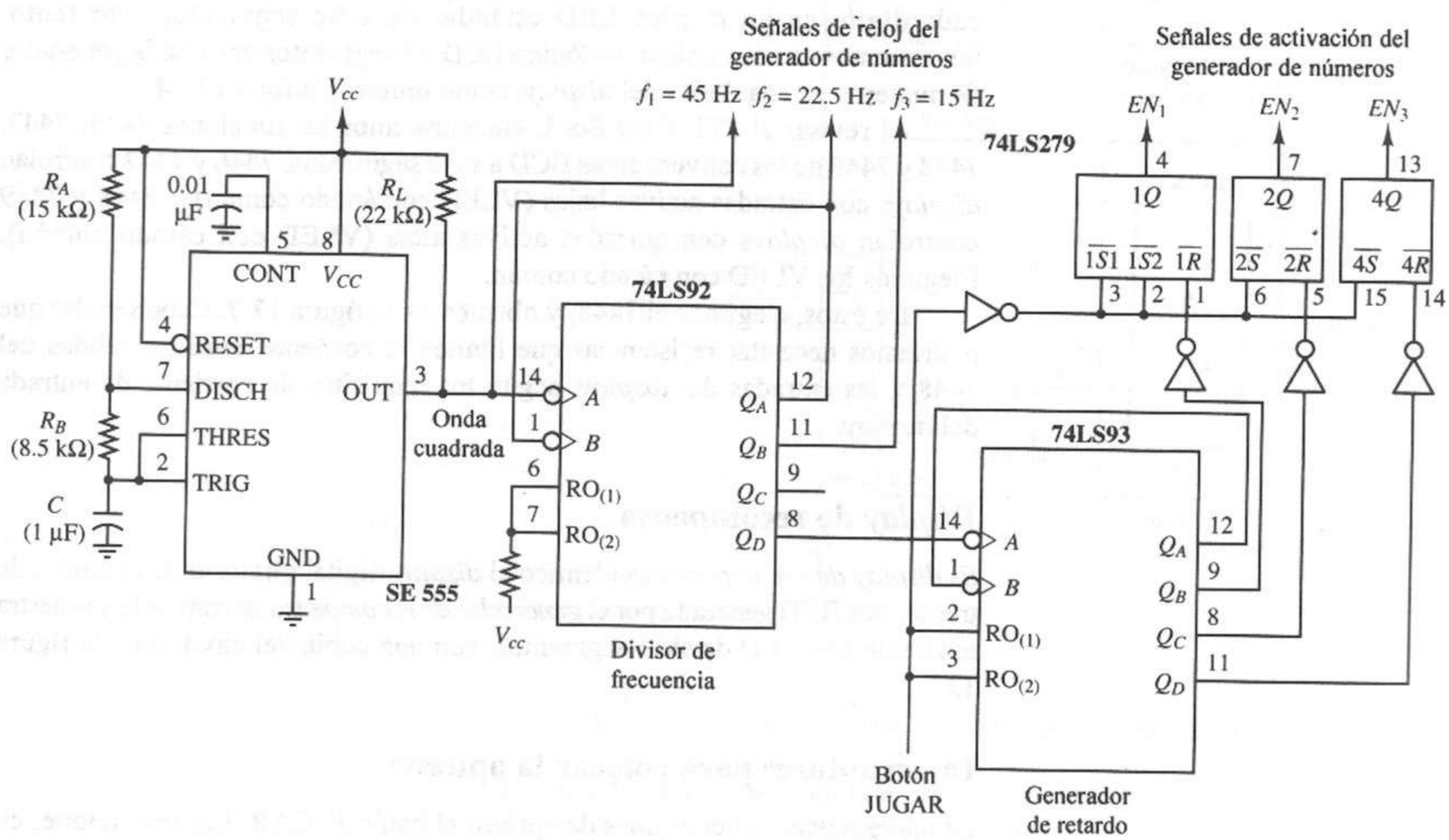


Figura 13.6 Módulo temporizador para el generador de números.

Obtenemos las señales de activación del contador EN_1, EN_2 y EN_3 a través de tres *latches* SR (SN74LS279). Los *latches* asumen el valor 1 al oprimir el botón JUGAR y después se restablecen a 0 en instantes distintos, después de que el botón JUGAR ha sido soltado. Para obtener instantes de restablecimiento diferentes, limpiaremos (con el conteo desactivado) un contador binario 74LS93 de 4 bits, mostrado en la figura 13.6, mientras se mantenga oprimido el botón JUGAR, y comenzará a contar cuando se suelte el botón JUGAR. Utilizaremos tres de las salidas del contador para restablecer los *latches*. $Q_B = 1$ (una cuenta de 2) desactiva EN_1, $Q_C = 1$ (una cuenta de 4) desactiva EN_2 y $Q_D = 1$ (una cuenta de 8) desactiva EN_3. El contador 74LS93 recibirá su entrada de reloj de la salida Q_D del 74LS92 utilizado para las señales de reloj del generador de números. La frecuencia de onda en la salida Q_D del 74LS92 es de

$$f = 45 \text{ Hz}/12 = 3.75 \text{ Hz}$$

Recordemos que el periodo de reloj es el inverso de la frecuencia de reloj, por lo que los tiempos de retardo para las tres señales de activación son

$$\text{Dígito 1} \quad \frac{1}{f/2} = 2 \div 3.75 \text{ Hz} = 0.533 \text{ s}$$
$$\text{Dígito 2} \quad \frac{1}{f/4} = 4 \div 3.75 \text{ Hz} = 1.06 \text{ s}$$
$$\text{Dígito 3} \quad \frac{1}{f/8} = 8 \div 3.75 \text{ Hz} = 2.13 \text{ s}$$

Display digital giratorio

La salida de cada generador de números aleatorios es un número binario de 4 bits que representa uno de los valores codificados en binario [1..7]. Exhibiremos cada dígito en un *display* LED estándar de siete segmentos. Por tanto, insertaremos un convertidor de código BCD a 7 segmentos entre cada generador de números y cada dígito del *display*, como muestra la figura 13.4.

Al revisar el *TTL Data Book*, encontraremos las funciones 7446, 7447, 7448 y 7449 de los convertidores BCD a siete segmentos. 7446 y 7447 controlan *displays* con entradas activas bajas (VLED con ánodo común) y 7448 y 7449 controlan *displays* con entradas activas altas (VLED con cátodo común). Elegimos los VLED con cátodo común.

De éstos, elegimos el 7448, y obtenemos la figura 13.7. Cabe señalar que podríamos necesitar resistencias que limiten la corriente entre las salidas del 7448 y las entradas del *display*, según los requisitos de corriente de entrada del *display*.

Display de recompensa

El *display de recompensa* es idéntico al *display* digital giratorio. Un número de tres dígitos BCD generado por el *generador de recompensa* se convierte y muestra mediante tres LED de siete segmentos, con una copia del circuito de la figura 13.7.

Interruptores para colocar la apuesta

La apuesta debe hacerse antes de oprimir el botón JUGAR. Las transiciones en los interruptores para colocar la apuesta no inician acción alguna. Por tanto, podemos utilizar interruptores simples DIP sin filtrado de rebotes. Un registro

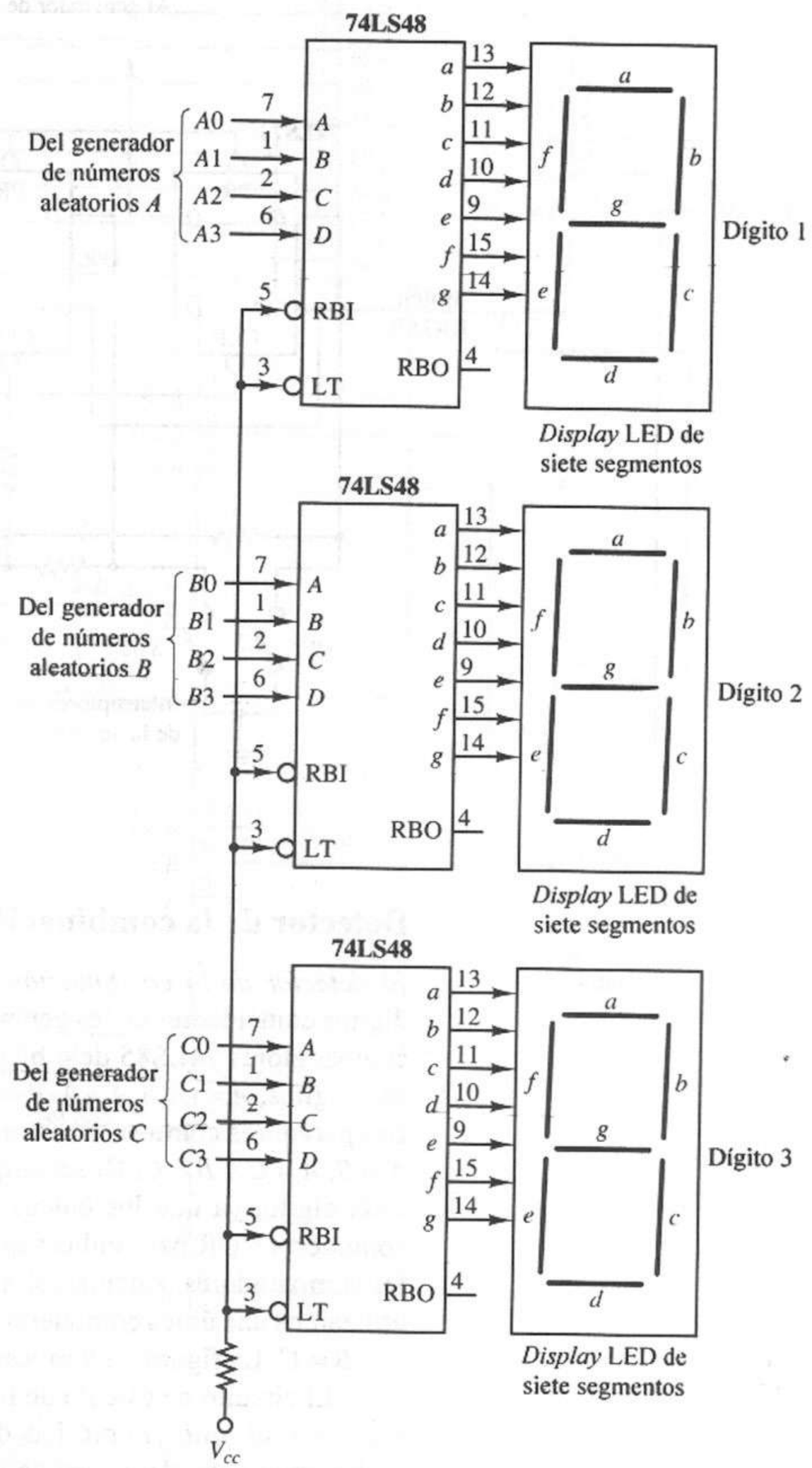


Figura 13.7 Circuito de interfaz con *display* de tres dígitos y siete segmentos.

de 2 bits fijará las posiciones de estos interruptores cuando se oprima el botón JUGAR, con objeto de evitar que la apuesta cambie una vez iniciado el juego. La salida de este registro se suministra como entrada al *generador de recompensa*. La figura 13.8 muestra el circuito.

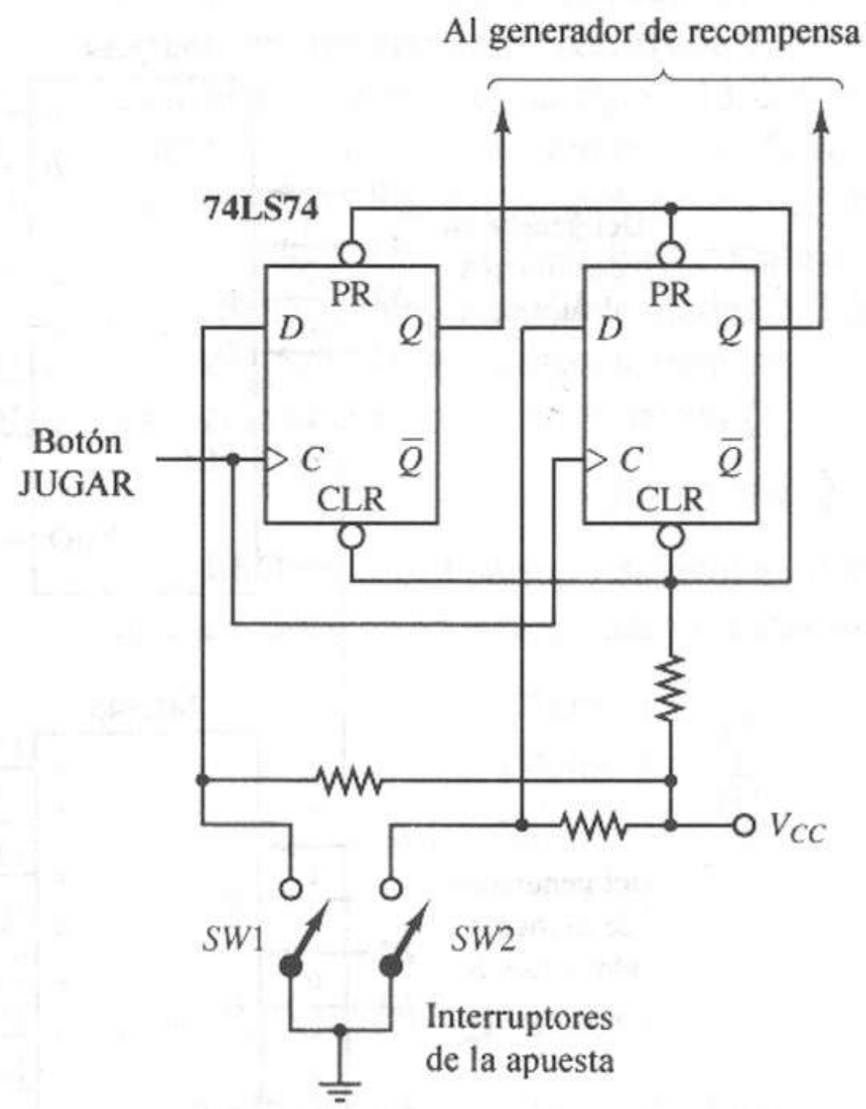


Figura 13.8 Interruptores y registro para la colocación de una apuesta.

Detector de la combinación ganadora

El *detector de la combinación ganadora* determina si obtenemos dos o tres dígitos coincidentes de los generadores de números. Hacemos esto mediante los comparadores 74LS85 de 4 bits para detectar los valores idénticos. Como hay tres dígitos, $A = (A_3A_2A_1A_0)$, $B = (B_3B_2B_1B_0)$ y $C = (C_3C_2C_1C_0)$, utilizaremos tres comparadores como se muestra en la figura 13.9 para detectar las condiciones $A = B$, $A = C$ y $B = C$. Observe que sólo hay que verificar los 3 bits menores de cada dígito, ya que los únicos dígitos válidos son del 1 a 7. Utilizamos una compuerta NOR para indicar que existe una concordancia en al menos uno de los comparadores. Además, si $A = B$ y $A = C$, esto implica que $B = C$. Por tanto, utilizamos una única compuerta AND de dos entradas para detectar la condición $A = B = C$. La figura 13.9 muestra el circuito completo.

El circuito de cálculo de la recompensa debe saber si coinciden dos o tres dígitos y el valor numérico del dígito concordante. Podemos utilizar un multiplexor para elegir uno de los números de entrada, si concuerda con uno o dos de los otros, y pasarlo al circuito de recompensa. Para esto, utilizamos un multiplexor 74LS157 cuádruple 2 a 1, con las dos entradas de 4 bits conectadas a las líneas de señal para los dígitos A y B, como muestra la figura 13.9. Si no hay dos dígitos concordantes, según indique la salida de la compuerta NOR, el multiplexor será desactivado por su entrada de control, G, anulando todas sus salidas. Si $B = C$, elegimos el dígito B usando la salida del comparador $B = C$ para controlar la línea de selección del multiplexor S. Si existe una concordancia

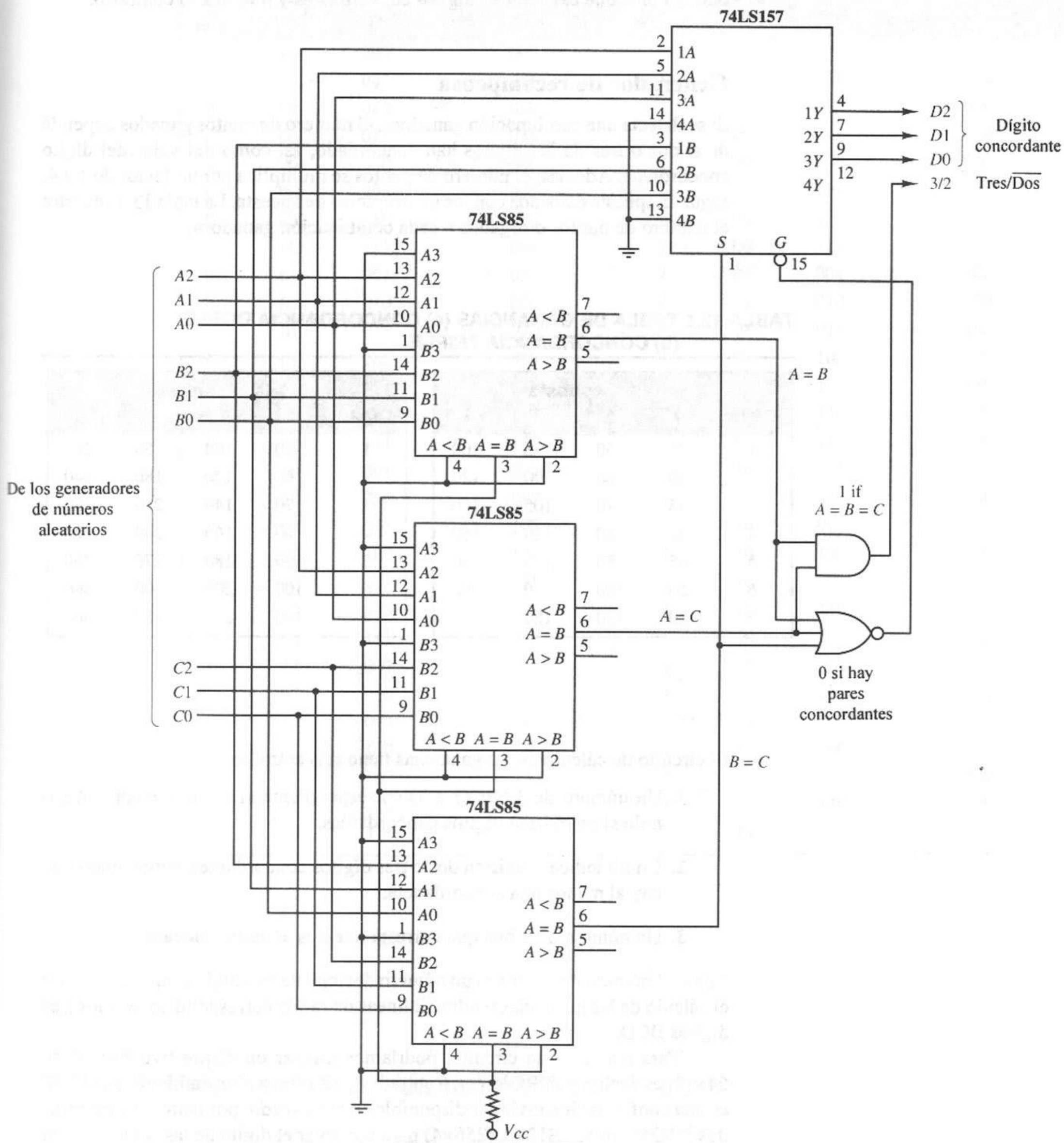


Figura 13.9 Circuito detector de una combinación ganadora.

y $B \neq C$, entonces $A = B$ o $A = C$, por lo que elegimos el dígito A. La salida de la compuerta AND también se lleva hasta el generador de recompensas. Observe que 1 indica que existen tres dígitos concordantes y 0 indica lo contrario.

Generador de recompensa

Si se detecta una combinación ganadora, el número de puntos ganados depende de si dos o tres de los dígitos han concordado, así como del valor del dígito concordante. Además, el número de puntos se multiplica por un factor de 1 a 4, según la apuesta colocada con los interruptores de apuesta. La tabla 13.1 muestra el número de puntos otorgados a cada combinación ganadora.

TABLA 13.1 TABLA DE GANANCIAS (a) CONCORDANCIA DOBLE (b) CONCORDANCIA TRIPLE

	Apuesta			
Dígito	1	2	3	4
1	25	50	75	100
2	30	60	90	120
3	35	70	105	140
4	40	80	120	160
5	45	90	135	180
6	50	100	150	200
7	60	120	180	240

	Apuesta			
Dígito	1	2	3	4
1	50	100	150	200
2	60	120	180	240
3	70	140	210	280
4	80	160	240	320
5	90	180	270	360
6	100	200	300	400
7	140	280	420	560

El circuito de cálculo de las ganancias tiene seis entradas:

1. Un número de 3 bits (1 a 7) correspondiente al dígito concordante, o nulo si no existen dígitos concordantes.
2. Un bit indica si existen dos o tres dígitos concordantes, suponiendo que hay al menos una concordancia.
3. Un número de 2 bits que corresponde a la apuesta colocada.

Como el número de puntos es un número decimal de tres dígitos, el circuito para el cálculo de las ganancias tendrá 12 líneas de salida correspondientes a los tres dígitos BCD.

Para realizar este circuito, podríamos utilizar un dispositivo PROM de 64×12, es decir, una PROM con 6 entradas y 12 salidas. Sin embargo, 64×12 no es una configuración estándar disponible en el mercado; por tanto, utilizaremos dos PROM, un 82LS129A (256×4) para controlar el dígito de las centenas y un 82LS135 (256×8) para controlar los dígitos de las decenas y las unidades. La figura 13.10 muestra el circuito generador de recompensas completo, con la asignación de señales a las entradas de dirección de la PROM de 8 bits y las salidas de la PROM. La tabla 13.2 enumera el contenido de la PROM.

TABLA 13.2 CONTENIDO DE LOS PROMS GENERADORES DE RECOMPENSAS

Entradas			Premio	Entradas			Premio
$A(5)$	$A(4\text{-}3)$	$A(2\text{-}0)$	$O(11\text{-}8)O(7\text{-}4)O(3\text{-}0)$	$A(5)$	$A(4\text{-}3)$	$A(2\text{-}0)$	$O(11\text{-}8)O(7\text{-}4)O(3\text{-}0)$
—	—	000	000				
0	00	001	025	1	00	001	050
0	00	010	030	1	00	010	060
0	00	011	035	1	00	011	070
0	00	100	040	1	00	100	080
0	00	101	045	1	00	101	090
0	00	110	050	1	00	110	100
0	00	111	060	1	00	111	140
0	01	001	050	1	01	001	100
0	01	010	060	1	01	010	120
0	01	011	070	1	01	011	140
0	01	100	080	1	01	100	160
0	01	101	090	1	01	101	180
0	01	110	100	1	01	110	200
0	01	111	120	1	01	111	280
0	10	001	075	1	10	001	150
0	10	010	090	1	10	010	180
0	10	011	105	1	10	011	210
0	10	100	120	1	10	100	240
0	10	101	135	1	10	101	270
0	10	110	150	1	10	110	300
0	10	111	180	1	10	111	420
0	11	001	100	1	11	001	200
0	11	010	120	1	11	010	240
0	11	011	140	1	11	011	280
0	11	100	160	1	11	100	320
0	11	101	180	1	11	101	360
0	11	110	200	1	11	110	400
0	11	111	240	1	11	111	560

13.2 Sistema de entrada automática sin llave

13.2.1 Definición del problema

Muchas veces, el propietario de un automóvil camina hacia éste y descubre que las llaves han quedado encerradas dentro del auto. Para resolver este problema, sería deseable un sistema de entrada sin llave que permita al propietario del

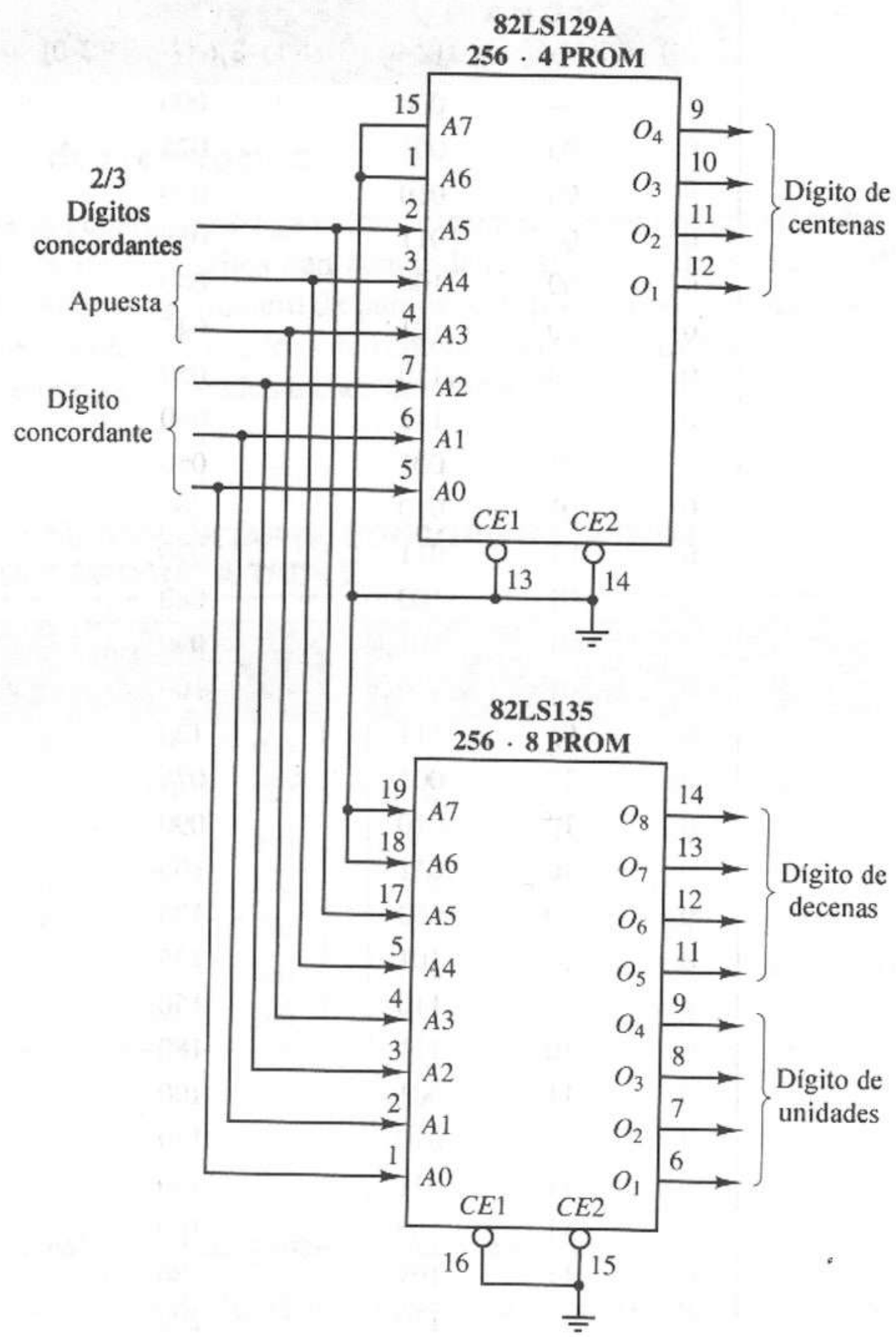


Figura 13.10 PROM generadora de recompensa.

automóvil introducir un código con una combinación numérica mediante un teclado de botones y así poder abrir la puerta del automóvil.

Por razones de seguridad, el diseño de tal sistema debe minimizar la posibilidad de que un ladrón descubra el código mediante prueba y error, como ocurre con muchas cerraduras de combinación. Podemos hacer varias cosas que dificulten la determinación del código correcto por experimentación. Uno consiste en hacer la longitud del código variable, de modo que el ladrón no sepa cuántos dígitos introducir. La tabla 23.3 muestra el número posible de combinaciones con cuatro teclas numéricas (1, 2, 3, 4), con códigos de longitud 4, 5, 6 y 7. La tabla también indica la probabilidad aproximada de adivinar la combinación correcta en un intento.

TABLA 13.3 COMBINACIONES PARA UN SISTEMA DE ENTRADA SIN LLAVE

Longitud del código	Número de combinaciones	Probabilidad aproximada de una estimación correcta en el primer intento	Tiempo para romper el código (1 código por minuto)
4	256	4/1,000	4.25 horas
5	1,024	1/1,000	2.1 días
6	4,096	1/5,000	1.7 semanas
7	16,384	3/50,000	6.8 semanas

Otro obstáculo para los ladrones sería desactivar de manera automática el sistema durante varios minutos si se realizan dos o tres intentos incorrectos por abrir la cerradura. Muchos ladrones no desean permanecer cerca y esperar para realizar otros intentos, aunque esto sería un inconveniente menor para el propietario del automóvil comparado con una llamada a un cerrajero. La tabla 13.3 enumera el tiempo estimado necesario para probar todos los códigos, si se prueba uno cada minuto, con una espera obligada de 3 minutos después de cada tres entradas incorrectas.

13.2.2 Necesidades del sistema

Operamos el sistema de entrada al automóvil sin llave mediante un teclado de cinco botones cerca de la manija externa de la puerta. Para no hacer demasiado complejo el proyecto, sólo utilizaremos cuatro botones numéricos (1,2,3,4). El código de entrada será una secuencia definida por el usuario de cuatro, cinco, seis o siete dígitos. También se dispondrá de un botón RESET, que puede utilizarse, por ejemplo, en caso de introducirse un código erróneo. Después de oprimir el botón RESET, la secuencia del código correcto deberá ser introducirse desde el comienzo.

Para que el propietario del automóvil pueda establecer un código de entrada que pueda recordar con facilidad, el interior del automóvil tendrá un tablero con el que el propietario podrá configurar interruptores que programen los dígitos de la combinación y la longitud del código.

El sistema tendrá una única salida, UNLOCK, la cual activará un mecanismo para poder abrir la puerta. Suponemos que esta señal electrónica tiene menos prioridad que la cerradura mecánica normal y el mecanismo de la llave. La figura 13.11 muestra un diagrama de bloques del sistema de entrada del auto sin llave. Contiene los subsistemas siguientes:

1. **Teclado para introducir el código.** Utilizaremos cuatro interruptores de botón que representarán los dígitos 1, 2, 3 y 4. Un quinto interruptor de botón activará la función RESET. Al oprimirse cada interruptor, generará un pulso alto lógico. Para garantizar que los pulsos sean limpios, cada interruptor tendrá un filtro de eliminación de rebotes. Las señales producidas por los cuatro interruptores de datos se envían a un circuito codificador, el cual generará un único pulso cada vez que se oprima uno de los cuatro botones numéricos, junto con un código binario de 2 bits que representa ese botón. El botón RESET generará un pulso para desactivar la unidad de control del sistema. Este botón

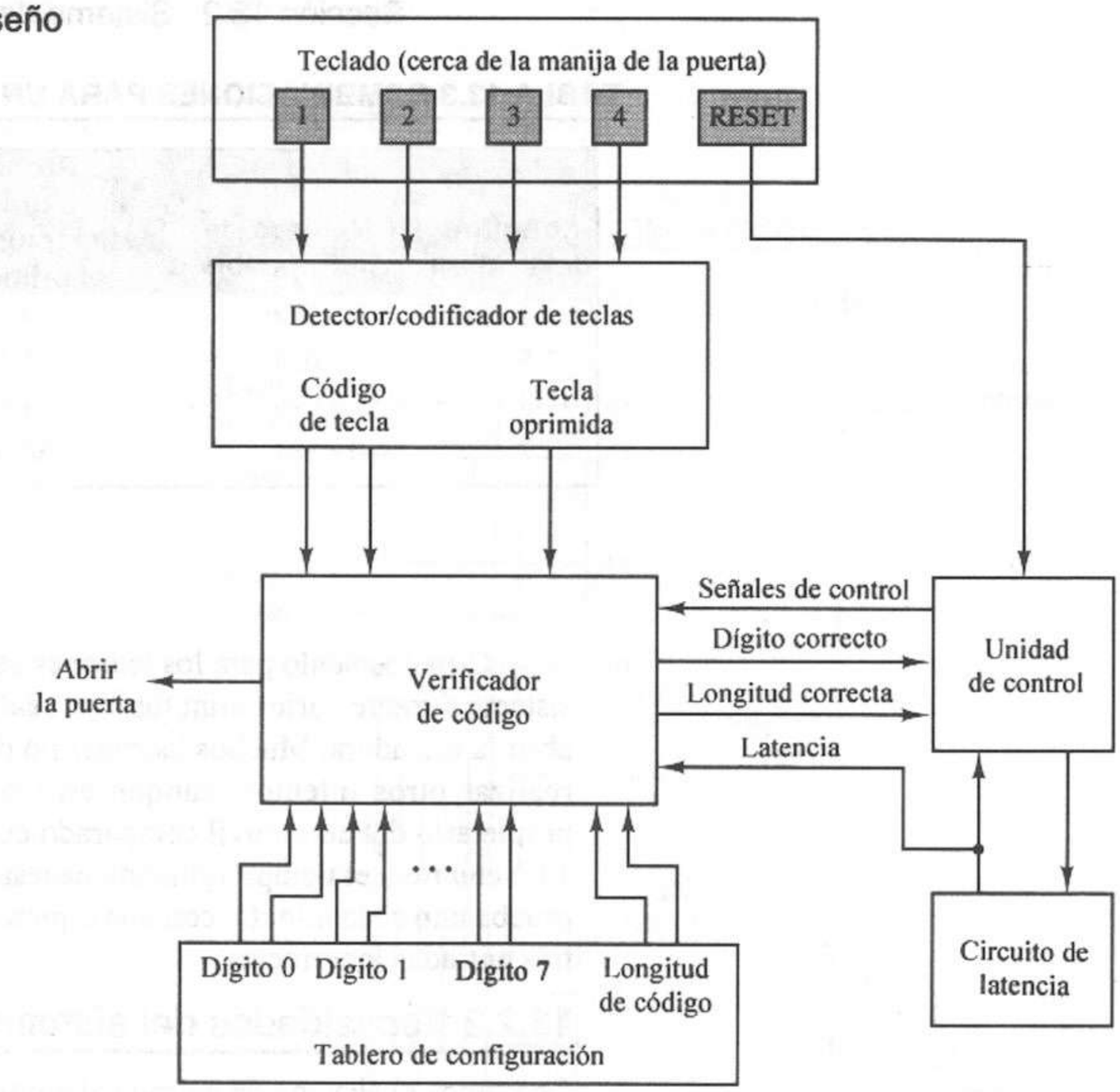
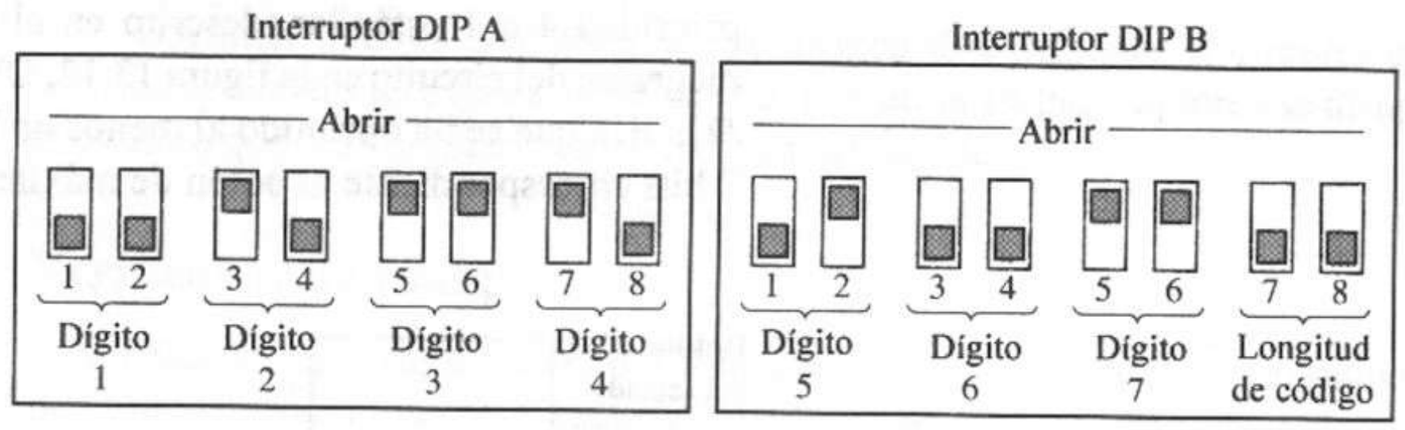


Figura 13.11 Diagrama de bloques del sistema de entrada a un auto sin llave.

será desactivado durante un intervalo "de latencia", cada vez que se hagan tres intentos incorrectos sucesivos por abrir la puerta.

2. **Tablero de configuración.** Proporcionaremos un tablero de 16 interruptores DIP, como se muestra en la figura 13.12a, para programar la longitud de la secuencia de código y los dígitos del código. Como la secuencia válida de dígitos puede tener una longitud de 4, 5, 6 o 7 dígitos, utilizaremos dos interruptores DIP, *B7* y *B8*, para establecer la longitud de la secuencia de código definida en la figura 13.12b. Cada dígito de la secuencia del código debe ser uno de los cuatro números 1, 2, 3 o 4. Por tanto, asignaremos dos interruptores DIP a cada dígito de la secuencia del código y los configuraremos como en la figura 13.12c. Como la longitud máxima del código es de 7 dígitos, necesitamos un total de 14 interruptores para definir cualquier secuencia de código posible. Enviaremos las señales de los 16 interruptores DIP al módulo verificador del código, el cual determina si se introdujo un código correcto.
3. **Verificador del código.** El verificador del código determina si una secuencia de dígitos introducidos mediante el teclado es la secuencia de código definida por los interruptores DIP del tablero de configuración. Este módulo requiere un contador que cuente el número de dígitos introducidos y un circuito que compare cada dígito con el par de interruptores correspondiente del tablero de activación. La unidad de control genera una señal UNLOCK si se detecta una



(a)

Longitud de código	Interruptor	Interruptor
4	Cerrado	cerrado
5	Cerrado	abierto
6	Abierto	cerrado
7	Abierto	abierto

(b)

Dígito	1er. interruptor	2o. interruptor
1	cerrado	cerrado
2	cerrado	abierto
3	abierto	cerrado
4	abierto	abierto

(c)

Figura 13.12 Tablero de configuración del sistema de entrada sin llave. **(a)** Interruptores DIP. **(b)** Parámetros de la longitud del código. **(c)** Códigos de dígitos.

secuencia de código correcta. La unidad de control genera una señal ERROR si cualquier dígito introducido es incorrecto o si se han introducido demasiados dígitos. Deberemos restablecer estas señales cuando la puerta se abra o se desactive la cerradura.

4. **Circuito de latencia**. La función del circuito de latencia es evitar que el sistema sea restablecido durante 3 minutos después de tres intentos consecutivos por abrir la cerradura. El circuito restablecerá el sistema en forma automática después de 3 minutos.
5. **Unidad de control**. La unidad de control proporcionará todas las señales de tiempo y control para los demás módulos. Al hacer esto, determinará si se está introduciendo una secuencia de código, si se necesita una señal *reset* después de introducir una secuencia de código incorrecta, o si hay que activar al circuito de latencia después de tres intentos incorrectos consecutivos por introducir la secuencia de código correcta.

13.2.3 Diseño lógico

Presentaremos el diseño del circuito lógico del sistema de entrada a un auto sin llave mediante la descripción de cada módulo identificado en las necesidades. Después combinaremos los módulos para crear el circuito completo. Los siguientes párrafos presentan el diseño de los módulos individuales.

Interfaz del teclado y codificador

La interfaz del teclado tiene cinco interruptores de botón, un circuito con filtrado de rebotes y un circuito codificador. Podemos utilizar un par de compuertas NAND con acoplamiento cruzado como circuito filtrador de rebotes para cada interruptor, como se mostró en la figura 13.3 para el ejemplo anterior. Para obtener el código de 2 bits cada botón podemos utilizar un codificador con

prioridad 4 a 1 estándar, descrito en el capítulo 4 (Fig. 4.19). Repetimos el diagrama del circuito en la figura 13.13. Observe que el circuito tiene tres salidas: D indica que se ha oprimido al menos un botón de dígito, y d_1d_0 es el código de 2 bits correspondiente al botón de máxima prioridad.

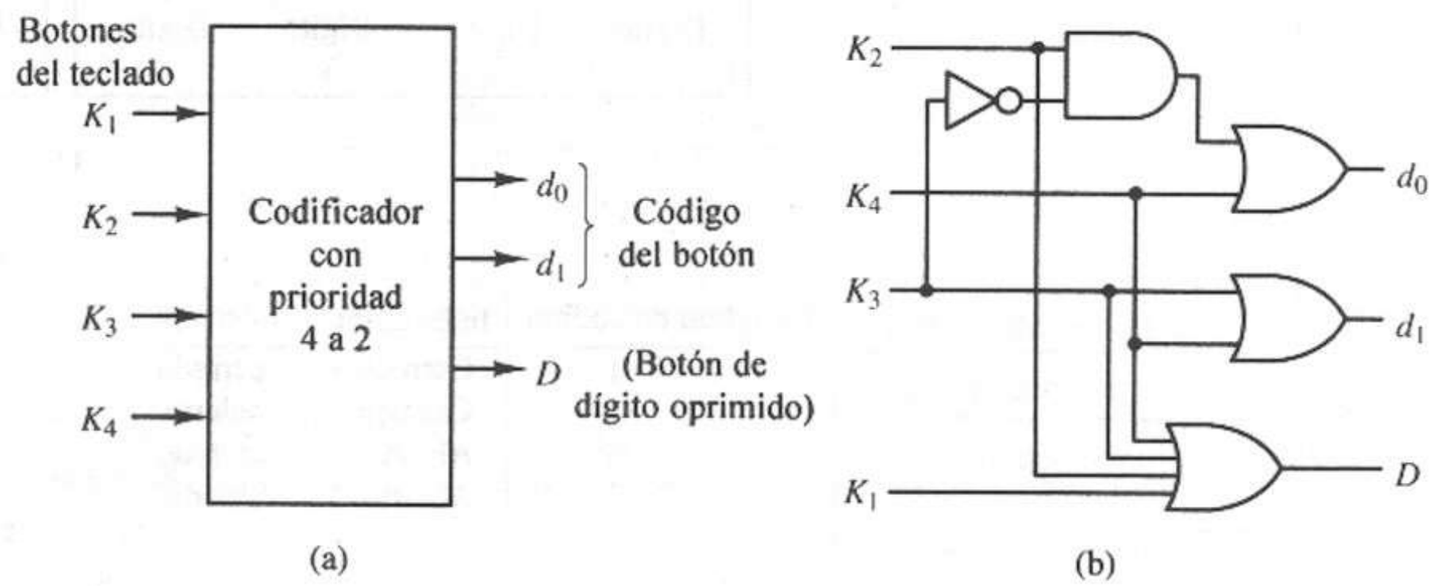


Figura 13.13 Codificador 4 a 1 con prioridad para el teclado de entrada de código. **(a)** Símbolo lógico. **(b)** Diagrama lógico.

Para funcionar correctamente, el resto del circuito esperará que la señal D sea alta y después baja conforme se oprima cada nuevo botón. Si se oprimen dos botones al mismo tiempo, sólo se detectará el botón con el número mayor.

Tablero de configuración

Los 16 interruptores DIP del tablero de configuración se configuran de antemano y supondremos que permanecen fijos durante el funcionamiento del sistema. En consecuencia, no necesitamos un circuito filtrador de rebotes. Configuramos los interruptores DIP como se muestra en la figura 13.14. Cada línea se lleva al valor lógico 1 cuando el interruptor está ABIERTO y al valor lógico 0 cuando está CERRADO.

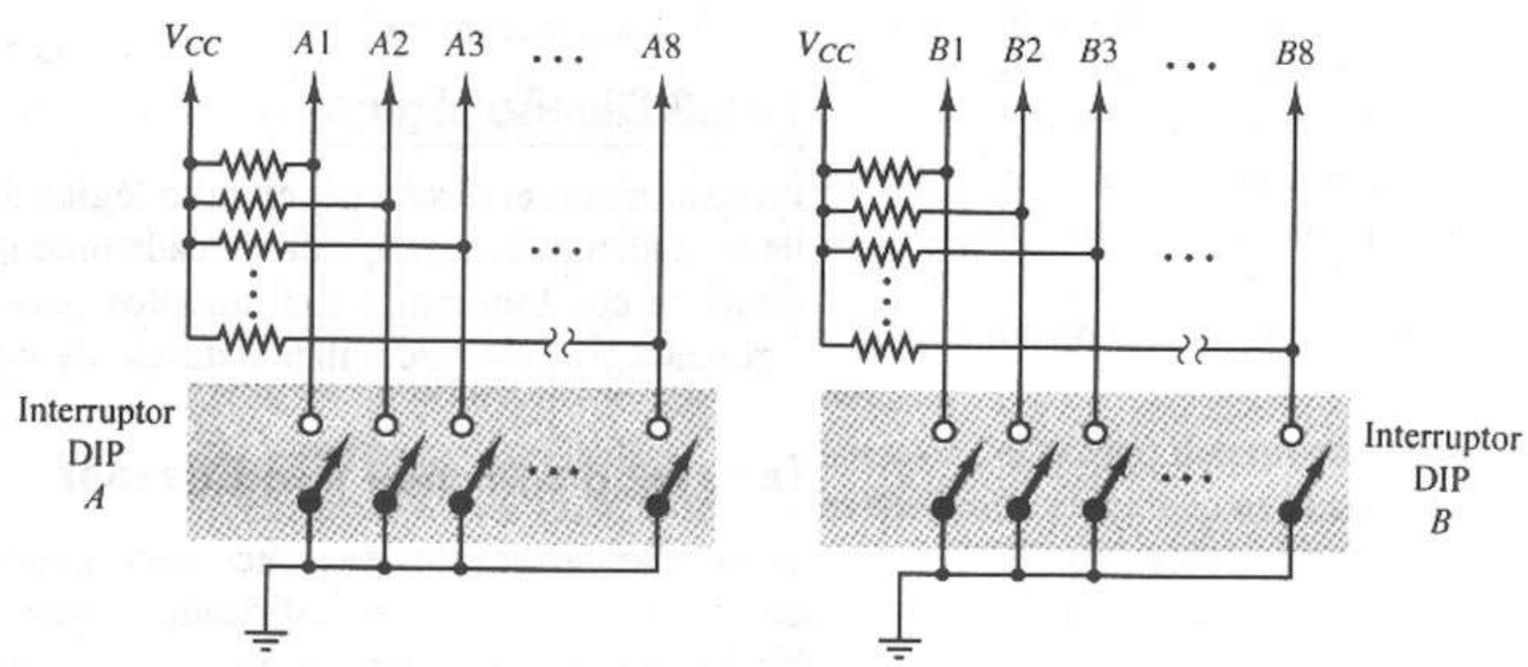


Figura 13.14 Interruptores DIP del tablero de configuración.

Los interruptores DIP *B7* y *B8* se dirigen directamente al circuito que verifica el número de dígitos introducidos. Los otros 14 interruptores se dirigen al circuito que los compara con el código introducido.

Verificador de código

El verificador de código realiza dos funciones. La primera es contar el número de dígitos introducidos en el teclado. Como la longitud máxima del código de entrada es de 7 dígitos, podemos utilizar un contador binario de 3 bits como contador de los dígitos. Incrementaremos el contador de dígitos en la transición descendente de cada pulso en INCD y lo limpiaremos cuando CLRD genere una señal RESET. En este ejemplo, elegimos un contador binario 74SL93 de 4 bits con *reset* asíncrono, conectado como se muestra en la figura 13.15.

El segundo 74LS93 de la figura 13.15 cuenta el número de intentos de introducción del código. Este contador se incrementa después de cada intento fallido en la transición descendente de un pulso en INCT y se limpia mediante un pulso en CLRT. En las salidas del contador, se obtiene un 1 en la salida *T* de la compuerta AND cuando la cuenta llega a 3; es decir, cuando $Q_B Q_A = 11$. Esto indica a la unidad de control que han ocurrido tres intentos fallidos por introducir el código.

La segunda función del verificador de código es comparar el codigo de 2 bits para el *n*–ésimo dígito de una secuencia de código introducida mediante el teclado con la configuración del par *n* de interruptores DIP del tablero de configuración. Necesitamos un circuito para elegir y guiar el par *n* de interruptores a un comparador que lo comparará con el código del dígito introducido. Puesto que hay siete parejas de interruptores, podemos usar un multiplexor dual 8 a 1 para esto. Utilizaremos dos módulos multiplexores 74LS151 8 a 1, como se muestra en la figura 13.15. Elegimos el par de interruptores mediante el contador de dígitos descrito anteriormente. El par 0 se elige inmediatamente después de que el contador se restablece a 0, el par 1 después de introducir el primer dígito, etc. Observe que el 74LS93 se incrementa en la transición descendente de cada pulso en INCD, pero la comparación se realiza mientras el pulso es alto.

Necesitamos dos circuitos de comparación, uno para comparar los dígitos introducidos con los pares correspondientes de interruptores DIP y el segundo para comparar el número de dígitos introducidos con los pares de interruptores que definen la longitud de la secuencia de código. En el primer caso podemos utilizar un comparador de 2 bits; para la longitud de la secuencia, podemos usar un comparador de 3 bits. En lugar de diseñar estos circuitos, utilizamos módulos comparadores 74LS85 de cuatro bits para cada uno, como en la figura 13.15. Para el primer 74LS85, las entradas superiores provienen del codificador del teclado y las inferiores del multiplexor de los interruptores DIP. Para el segundo 74LS85, las entradas superiores provienen del contador de dígitos, mientras que las entradas inferiores provienen de los interruptores DIP *B7* y *B8*.

Circuito de latencia

El circuito de latencia debe inhibir el funcionamiento del sistema durante un periodo de 3 minutos después de tres intentos incorrectos por introducir la combinación. Haremos esto mediante un módulo temporizador 555, configurado como en la figura 13.16 para una operación *monoestable* (*one-shot*). El 555 será disparado por un pulso activo bajo desde la unidad de control según la señal

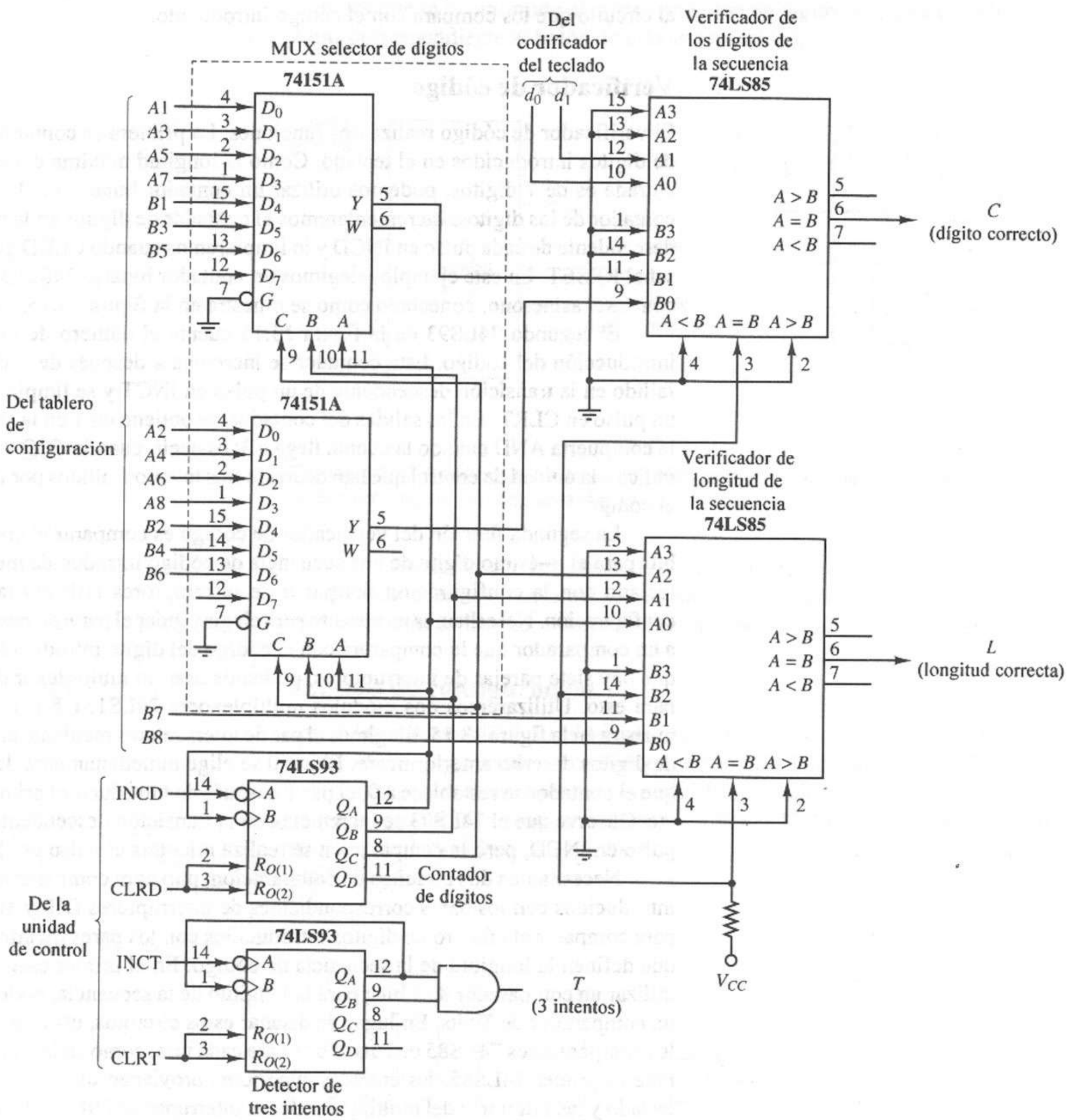


Figura 13.15 Diagrama lógico del verificador de código.

señal $\overline{SLP}$ siempre que se detecte una tercera secuencia de introducción de código incorrecta consecutiva, generando una señal SLEEP en forma de pulso con duración de 3 minutos que evitará el restablecimiento de la unidad de control.

Recordemos del capítulo 6 que el ancho de pulso de salida del 555 está dado por

$$t_W = 1.1(R_A)(C) \quad \text{s} \tag{13.2}$$

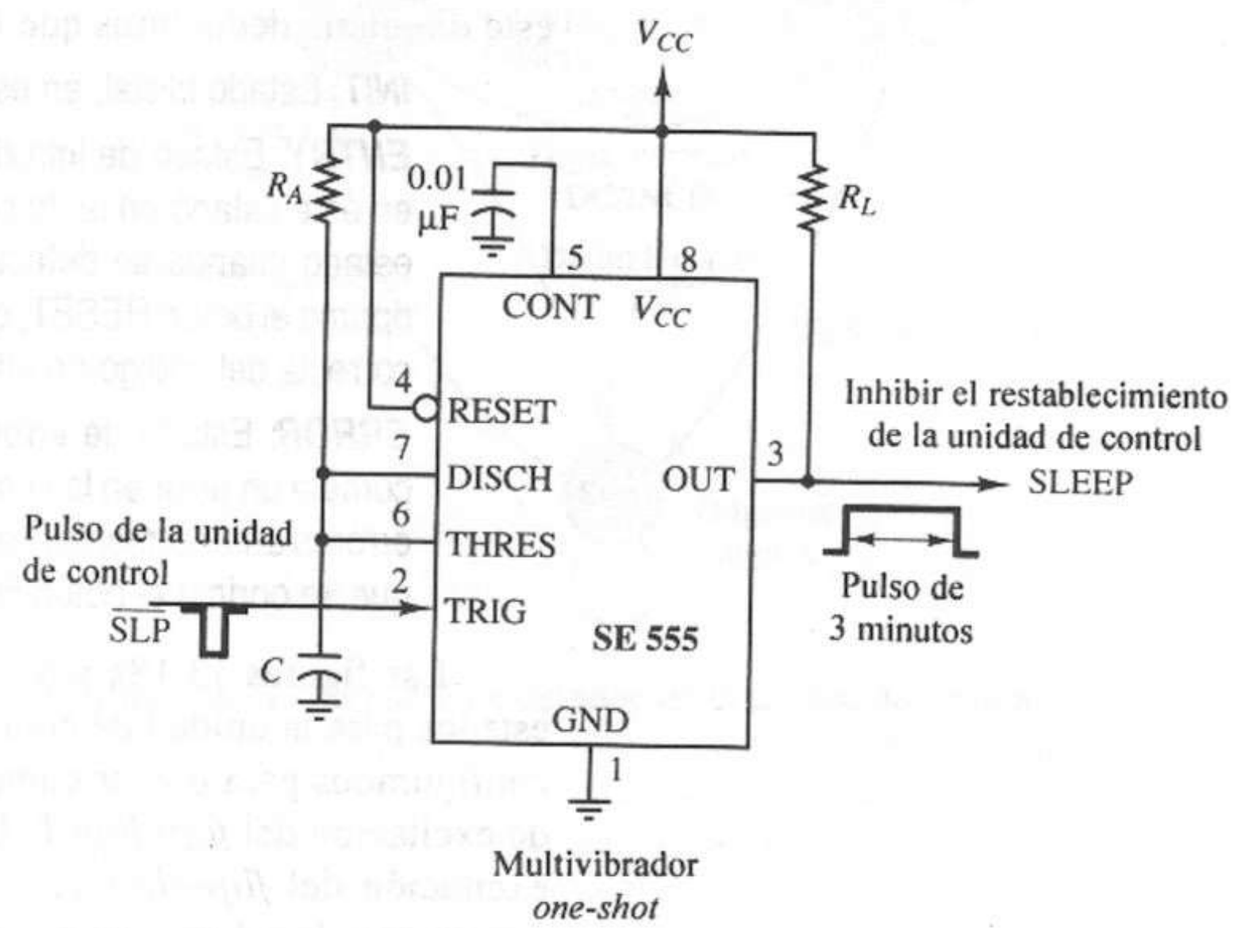


Figura 13.16 Diagrama lógico "*one-shot*" del circuito de latencia.

Por tanto, los valores $R_A = 3\ M\Omega$ y $C = 60\ \mu F$ producen un pulso con una duración aproximada de 3 minutos.

Unidad de control

En respuesta a las entradas desde el teclado, la unidad de control se encarga de determinar el momento de activar el mecanismo UNLOCK, el momento de activar el circuito SLEEP, y el momento de restablecer el sistema.

Como hemos visto, el codificador del teclado produce un pulso en la señal *D* siempre que se oprima uno de los cuatro botones de dígito, y el botón RESET produce un pulso en la señal *R*. Como éstas son las señales primarias que inician acciones en el sistema, diseñaremos la unidad de control como un circuito secuencial asíncrono de modo pulso, según lo descrito en el capítulo 10.

Al generar los pulsos en las señales *D* y *R*, otras tres condiciones determinan lo que debe hacer la unidad de control. Estas tres condiciones están representadas por las líneas de señal *C, L* y *T* como sigue:

> $C = 1$ si el dígito introducido en ese momento es correcto, y 0 si el dígito actual es incorrecto.
>
> $L = 1$ si el número de dígitos introducidos es igual a la longitud de la secuencia de código de entrada, y 0 en caso contrario.
>
> $T = 1$ si han ocurrido tres intentos anteriores por introducir la secuencia de código, y 0 en caso contrario.

La figura 13.15 muestra que la señal *C* es la salida del comparador 74LS85, el cual verifica los dígitos correctos, la señal *L* es la salida del 74LS85 que compara

la salida del contador de dígitos con la longitud del código seleccionado, y T se obtiene a partir del contador del "detector de tres intentos".

Como vimos en el capítulo 10, iniciamos el diseño de la unidad de control desarrollando un diagrama de estados, el cual aparece en la figura 13.17. De este diagrama deducimos que la unidad de control tiene tres estados:

INIT: Estado inicial, en espera de la introducción del primer dígito.

ENTRY: Estado de introducción de dígitos; la unidad de control permanecerá en este estado en tanto se introduzcan los dígitos correctos. Se saldrá de este estado cuando se detecte un error en la introducción del código, cuando se oprima el botón RESET, o cuando se haya introducido una secuencia completa correcta del código de entrada.

ERROR: Estado de error; la unidad de control entrará en este estado si se comete un error en la secuencia del código de entrada y si no han ocurrido tres errores anteriormente. La unidad de control permanecerá en este estado hasta que se oprima el botón RESET, momento en que regresará al estado *INIT*.

Las figuras 13.18a y b muestran la tabla de estados y la tabla binaria de estados para la unidad de control, respectivamente. Utilizaremos *flip–flops* JK configurados para operar como *flip–flops* T. La figura 13.18c muestra la tabla de excitación del *flip–flop* T. De esta tabla podemos deducir las ecuaciones de excitación del *flip–flop* T, y de la tabla de transición podemos deducir las ecuaciones de salida para el circuito secuencial de modo pulso.

$$
\begin{aligned}
T_1 &= (D\bar{C}\bar{L})\bar{y}_1 + (R\bar{T})y_1 \\
T_2 &= (DC\bar{L})\bar{y}_1\bar{y}_2 + (DCL)\bar{y}_1 + (D\bar{C}\bar{T} + D\bar{C}T + R\bar{T} + RT)y_2 \\
&= DC\bar{y}_1\bar{y}_2 + DCL\bar{y}_1 + D\bar{C}y_2 + Ry_2 \\
UNLK &= (DCL)\bar{y}_1 \\
SLP &= (D\bar{C}T)\bar{y}_1 + RT \\
INCD &= (DC\bar{L})\bar{y}_1 \\
INCT &= R\bar{T} \\
CLRD &= (DCL)\bar{y}_1 + R\bar{T} \\
CLRT &= (DCL)\bar{y}_1
\end{aligned}
$$

La figura 13.19 muestra el diagrama lógico para la unidad de control.

13.3 Controlador de tráfico de un carril

En muchos lugares, el tráfico de automóviles en doble sentido debe pasar por un tramo de un solo carril, como en los puentes angostos del campo, los caminos en reparación y otras calles estrechas. Como se muestra en la figura 13.20, el camino sencillo une por lo general dos segmentos de camino de doble sentido. Para controlar el tráfico en doble sentido en este camino sencillo necesitamos semáforos especiales en cada extremo del camino de un carril que permitan al tráfico fluir en una dirección durante un periodo de tiempo y después detenerlo para que el tráfico fluya en la otra dirección, de manera alternada. En cada cambio de dirección, el controlador del semáforo debe detener el tráfico en una dirección y esperar hasta que el carril esté libre antes de permitir que el tráfico avance en

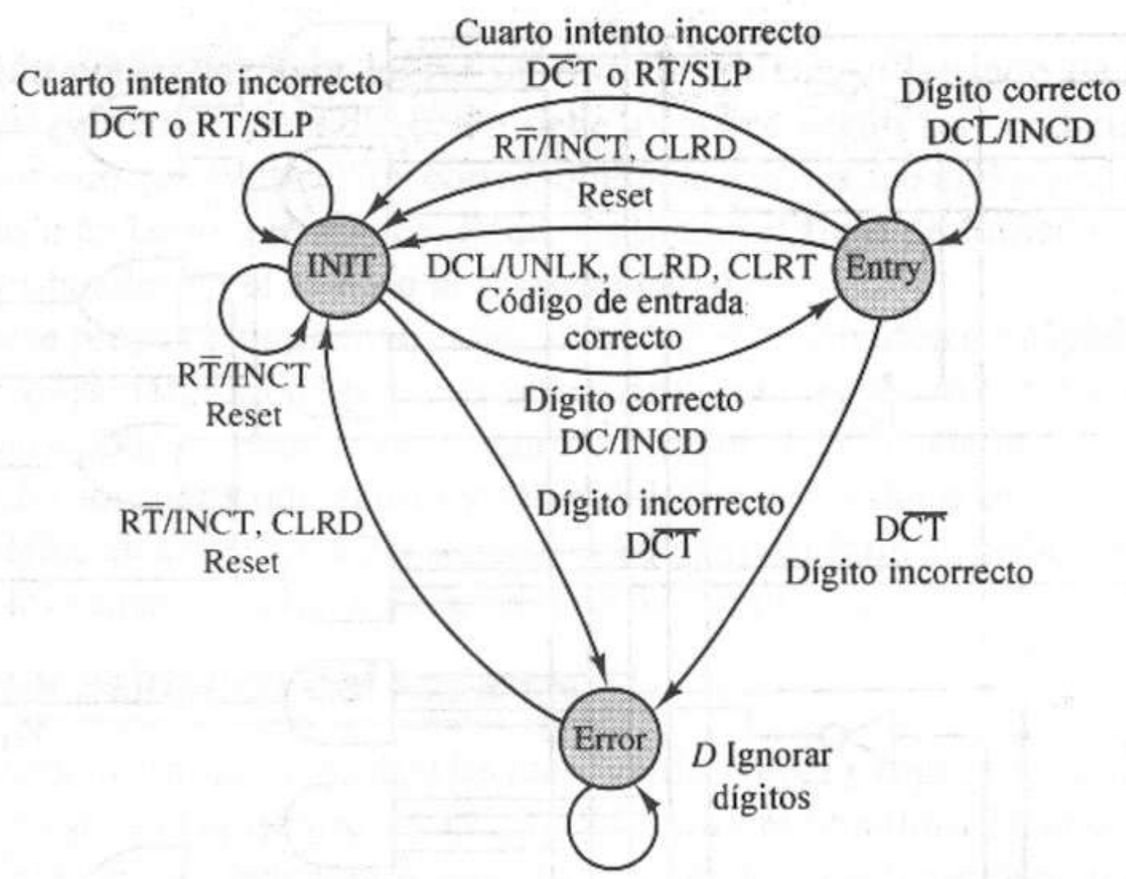


Figura 13.17 Diagrama de estados de la unidad de control.

Estado:	$DC\overline{L}$	DCL	$D\overline{C}\overline{T}$	$D\overline{C}T$	$R\overline{T}$	RT
INIT = 0	1/INCD	—	2	0/SLP	0/INCT, CLRD	0/SLP
ENTRY = 1	1/INCD	0/UNLK, CLRD, CLRT	2	0/SLP	0/INCT, CLRD	0/SLP
ERROR = 2	2	2	2	2	0/INCT, CLRD	—

Estado siguiente/salidas

(a)

$y_1 y_2$	$DC\overline{L}$	DCL	$D\overline{C}\overline{T}$	$D\overline{C}T$	$R\overline{T}$	RT
INIT = 00	01	—	10	00	00	00
ENTRY = 01	01	00	10	00	00	00
11	—	—	—	—	—	—
ERROR = 10	10	10	10	10	00	—

$Y_1 Y_2$

(b)

$y_1 y_2$	$DC\overline{L}$	DCL	$D\overline{C}\overline{T}$	$D\overline{C}T$	$R\overline{T}$	RT
INIT = 00	01	—	10	00	00	00
ENTRY = 01	00	01	11	01	01	01
11	—	—	—	—	—	—
ERROR = 10	00	00	00	00	10	—

$T_1 T_2$

(c)

Figura 13.18 Tablas de estados, de transición y de excitación de la unidad de control. **(a)** Tabla de estados. **(b)** Tabla de estados binarios. *(c) Tabla de excitación del flip-flop.*

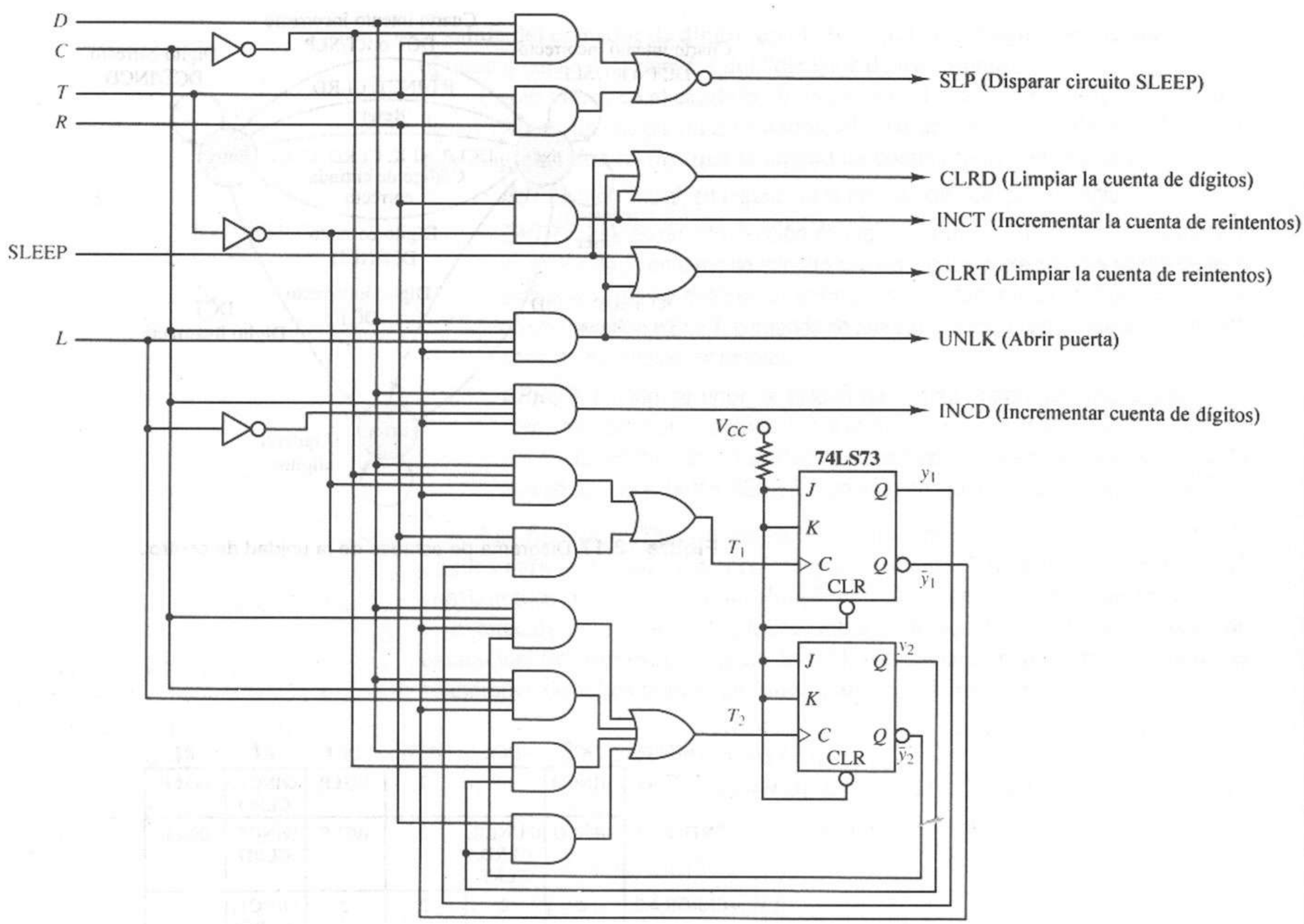


Figura 13.19 Diagrama lógico de la unidad de control.

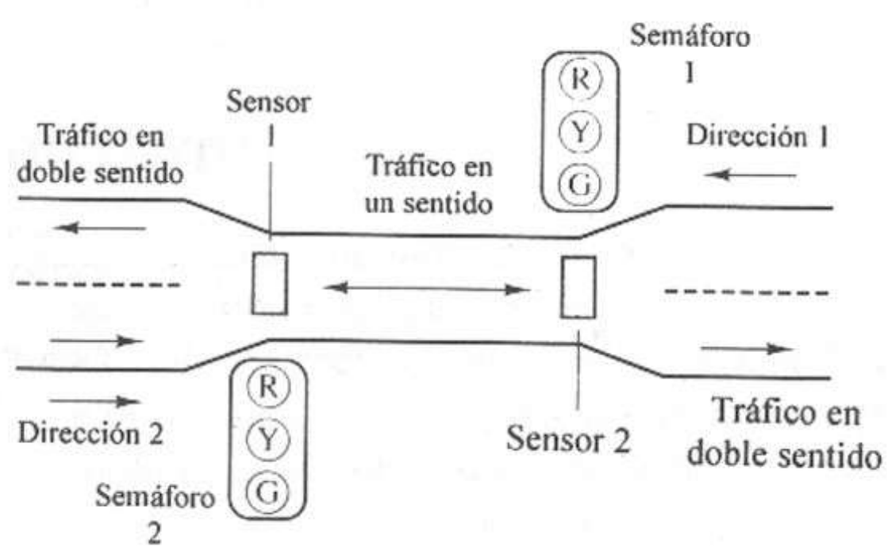


Figura 13.20 Tráfico de doble sentido en un camino de un carril.

la dirección contraria. Para lograr un tráfico óptimo, el periodo de tiempo asignado al tráfico en cada dirección debe ajustarse según las condiciones de tráfico, de modo que la dirección correspondiente a un tráfico más pesado reciba un periodo más largo que el otro. Podemos medir el flujo de tráfico mediante sensores instalados en el camino al final del carril.

En este proyecto diseñaremos un controlador de semáforos adaptable para coordinar los semáforos en los dos extremos de un camino de un carril y permitir el tráfico en ambos sentidos. Colocaremos un sensor en cada extremo del camino para detectar los autos que entran y salen del camino. Asignaremos tiempo al flujo de tráfico en cada dirección según la medición del flujo de tráfico obtenida mediante los sensores durante cada periodo de 5 minutos.

13.3.1 Necesidades del sistema

El controlador de tráfico controlará las luces verde, ámbar y roja de dos semáforos ($G1$, $Y1$ y $R1$ para el semáforo 1 y $G2$, $Y2$ y $R2$ para el semáforo 2), uno en cada extremo del camino. Suponemos que cada una de las seis luces tiene una línea de control ON/OFF aparte. Las entradas al controlador de tráfico incluyen las señales de dos sensores, $S1$ y $S2$, colocados en cada extremo del camino. Cada sensor genera un pulso siempre que cruza un auto. También se dispone de un botón RESET para inicializar el controlador.

La principal función del controlador consiste en determinar cuándo cambiar los semáforos de un color al siguiente. Para los autos que se mueven en la dirección 1, $G1$ estará activo durante un tiempo T_1, que será calculado de nuevo cada 5 minutos según el flujo del tráfico en cada dirección. Después del instante T_1, se activará la luz ámbar $Y1$ por una única unidad de tiempo T_Y (utilizaremos 10 segundos como la unidad básica de tiempo para este ejemplo), y después se encenderá la luz roja $R1$ hasta que el controlador esté listo para activar $G1$ de nuevo. La figura 13.21 ilustra este patrón de tiempos.

Para los autos que se mueven en la dirección 2, la luz $G2$ no se activa sino hasta que el último auto que circula en la dirección 1 ha salido del camino. El número de autos que permanecen en el camino se puede determinar comparando el número de autos que entran en él, indicado mediante un sensor, con el número de autos que salen del camino, indicado por el otro sensor. Cuando la diferencia

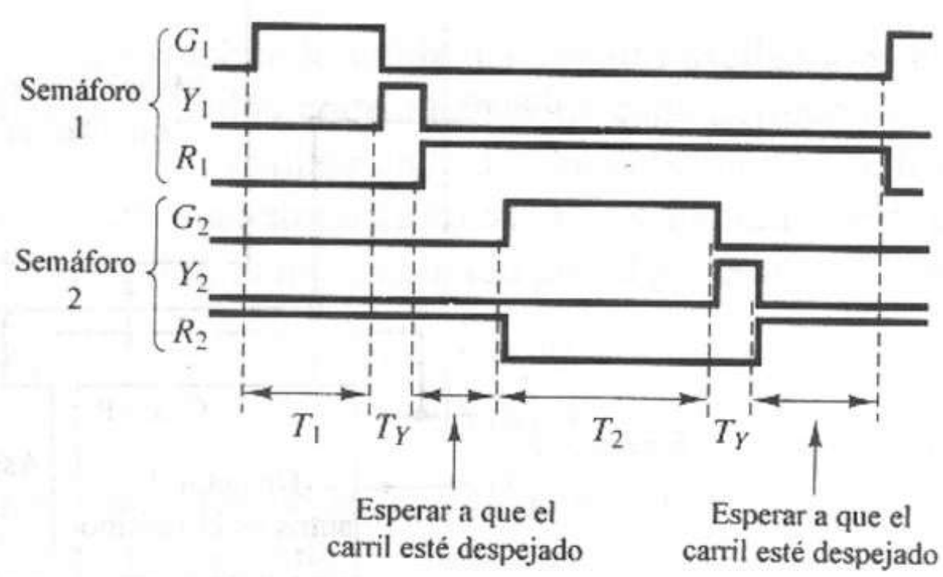


Figura 13.21 Diagrama de tiempos del controlador de tráfico.

entre ambos números es cero, suponemos que el camino está vacío. Calculamos la duración de la luz verde $G2$, T_2, como $T_2 = T_{tot} - T_1$, donde T_{tot} es una cantidad total de tiempo de luz verde. T_{tot} se dividirá entre las direcciones 1 y 2 según el flujo de tráfico relativo en cada dirección.

La figura 13.22 presenta un diagrama de bloques del controlador de semáforos. Los principales componentes del controlador son un oscilador de base de tiempo, un contador para determinar si permanecen autos en el camino, un contador de tráfico para medir el flujo de tráfico relativo en ambas direcciones, un circuito para calcular las asignaciones de tiempo de duración de la luz verde para ambas direcciones, y una unidad de control de la máquina de estados. A continuación describimos de manera breve las funciones de estos módulos.

1. **Oscilador de base de tiempo**. Este oscilador generará una señal de reloj que será utilizada para calcular los momentos en que las luces se alternarán. Supondremos que el periodo de 10 segundos para la luz ámbar será el evento más corto de este sistema. Calcularemos los demás tiempos de alternancia como múltiplos de 10 segundos. Así, utilizaremos una señal de reloj con un periodo de 10 segundos. Calcularemos la cantidad de tiempo de luz verde asignada a cada dirección cada 5 minutos. Por tanto, el oscilador incrementará un contador que servirá para generar un pulso cada 5 minutos.
2. **Contador de autos en el camino**. Para determinar si quedan autos en el camino antes de activar una luz verde utilizaremos un contador que calcule la diferencia entre el número de autos que entran en el camino, N_E, y el número de autos que salen del camino, N_L. Supondremos que el camino está libre si $N_E - N_L = 0$. El número de autos que entran en el camino se determina contando los pulsos de un sensor, mientras que el número de autos que salen se determina contando los pulsos del otro sensor. Como la única condición de interés es $N_E - N_L = 0$, no necesitamos contar realmente. Por tanto, utilizaremos un contador arriba/abajo que se incrementará mediante los pulsos del sensor S_1 y se decrementará mediante los pulsos del sensor S_2. Una señal de salida del contador indicará la condición $N_E - N_L = 0$.

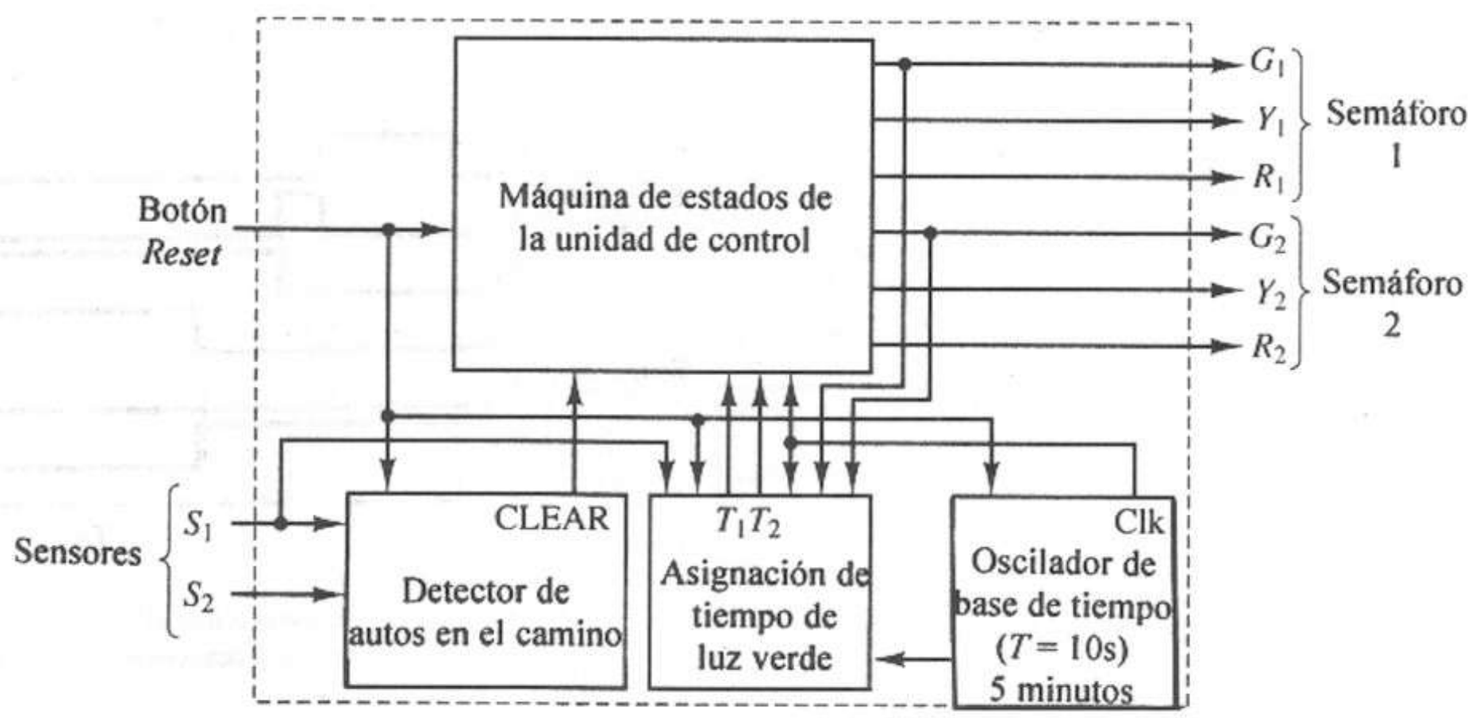


Figura 13.22 Diagrama de bloques del controlador de tráfico.

3. **Contador de tráfico.** Para determinar la cantidad relativa de tiempo de luz verde asignada a cada dirección utilizaremos un contador que calcule la diferencia entre el número de autos que circulan en el camino en cada dirección. Como en el caso del contador de autos en el camino, el contador de tráfico se incrementa con los autos que circulan en una dirección y se decrementa con los autos que circulan en la dirección opuesta. Utilizaremos los pulsos de S_1 en ambos casos. Muestrearemos el contador cada 5 minutos, según la indicación de un pulso del oscilador de base de tiempo, después de lo cual el contador restablecerá su valor en cero para comenzar el siguiente periodo de 5 minutos.
4. **Asignación de tiempo de luz verde.** Este módulo calcula de nuevo la duración de las luces verdes T_1 y T_2 al final de cada periodo de 5 minutos, con base en la salida del contador de tráfico. Si D_1 es el contador de tráfico en la dirección 1 y D_2 es el contador en la dirección 2, T_1 se incrementa si $D_1 - D_2 > 0$ y se decrementa si $D_1 - D_2 < 0$. Calcularemos T_2 como $T_{tot} - T_1$. Utilizaremos valores límite para garantizar que ni T_1 ni T_2 bajen de un periodo mínimo de 40 segundos para evitar una obstrucción en el flujo de tráfico en una dirección.
5. **Unidad de control de la máquina de estados.** La unidad de control de la máquina de estados coordinará el funcionamiento del controlador del semáforo y generará las señales ON/OFF para las seis lámparas, formando un ciclo de acuerdo con el diagrama de tiempos de la figura 13.21, de modo que los instantes de conmutación estén basados en las salidas del detector de autos en el camino y el módulo de asignación de tiempo de luz verde.

13.3.2 Diseño lógico

En esta sección desarrollaremos el diseño de los módulos individuales ya descritos. Después conectaremos estos módulos entre sí para completar el diseño del sistema.

Generador de base de tiempo

Una señal de reloj con un periodo de 10 segundos proporcionará la base de tiempo para el controlador. Esta señal de reloj será generada mediante un temporizador 555 que operará en modo multivibrador astable, como vimos antes en la figura 13.6 para el ejemplo de la máquina tragamonedas. La ecuación 13.2 implica que podemos obtener un periodo de 10 segundos si elegimos los siguientes valores de la resistencia y el condensador:

$$R_A = 200k\Omega$$
$$R_B = 200k\Omega$$
$$C = 24\,\mu F$$

La figura 13.23 muestra que la salida del circuito oscilador es la señal *CLK*.

Necesitamos un pulso corto al final de cada periodo de 5 minutos para señalar que es tiempo de realizar un muestreo del contador de tráfico y calcular de nuevo la asignación de tiempo de luz verde. Deduciremos este pulso mediante un contador binario simple incrementado por el generador de reloj. Como

$$\begin{aligned} 5 \text{ min} &= 5 \times 60 \text{ s} \\ &= 30 \times 10 \text{ s} \end{aligned}$$

usaremos un contador módulo 30. El 74LS390 es un contador dual asíncrono de décadas de 4 bits que podemos utilizar para este fin, como se muestra en la figura 13.23a. En este ejemplo, el contador 1 se incrementará una vez cada 10 segundos y el contador 2 se incrementará cuando el contador 1 cambie de 9 →

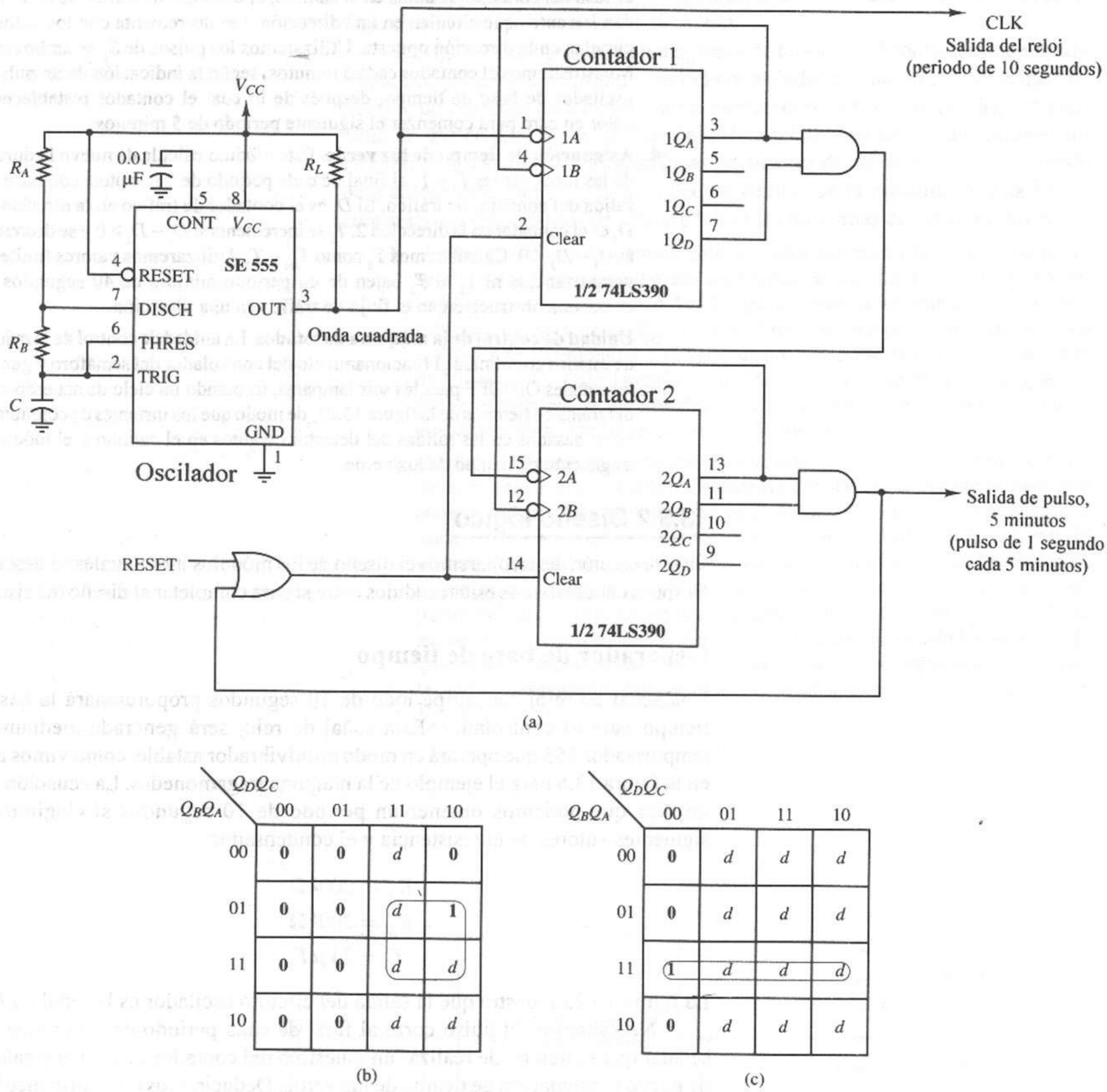


Figura 13.23 Diseño de circuito del generador de base de tiempo. **(a)** Diagrama lógico. **(b)** Contador 1 = 9. **(c)** Contador 2 = 3.

0. En relación con el mapa K de la figura 13.23b, como el contador 1 nunca excede 9, el contador 2 debe incrementarse según la condición

$$Q_A \cdot Q_D = 1$$

De igual manera, el contador 2 nunca excede 3. Por tanto, según el mapa K de la figura 13.23c, ambos contadores deben restablecerse para tener la condición del contador 2

$$Q_A \cdot Q_B = 1$$

Además, ambos contadores serán desactivados por la señal maestra RESET. La figura 13.23a muestra el circuito generador de base de tiempo completo.

Contador de autos en el camino

Como describimos anteriormente, el camino estará despejado siempre que el número de autos que salen del camino (N_L) sea igual al número de autos que entran en él (N_E). Para detectar esta condición, utilizaremos un contador binario arriba/abajo como sigue. Generaremos pulsos desde los sensores S_1 y S_2 cada vez que un auto entre o salga del camino. Para el tráfico en la dirección 1, los pulsos de S_1 indican autos que entran en el camino, mientras que para la dirección 2 indican autos que salen del camino, y viceversa para el sensor S_2. Como sólo importa la diferencia entre los autos que entran y salen, utilizaremos los pulsos de S_1 para incrementar el contador y los pulsos de S_2 para decrementarlo. Cada vez que la cuenta sea igual a cero, el contador se habrá incrementado y decrementado un mismo número de veces; es decir, $N_L - N_E = 0$, lo cual indica que el camino está despejado.

El contador binario arriba/abajo utilizado para este módulo debe tener un número suficiente de bits para contar el número máximo de autos que pueden entrar en el camino sin salir de él; es decir, para calcular el máximo valor esperado de $N_L - N_E$. En este ejemplo supondremos que basta un contador binario de 4 bits; es decir, no habrá más de 15 autos en el camino en cualquier instante dado. El 74LS193 es un contador binario de 4 bits arriba/abajo con entradas de reloj independientes para contar hacia arriba y hacia abajo. La figura 13.24 muestra su configuración, de modo que la entrada de reloj UP sea controlada por pulsos desde el sensor S_1 y la entrada de reloj DOWN sea controlada por S_2. Una compuerta NOR de 4 entradas detectará una cuenta igual a cero y producirá una salida de 1 lógico, lo que indicará que $N_E = N_L$, en cuyo caso todos los autos que han entrado al camino lo han dejado.

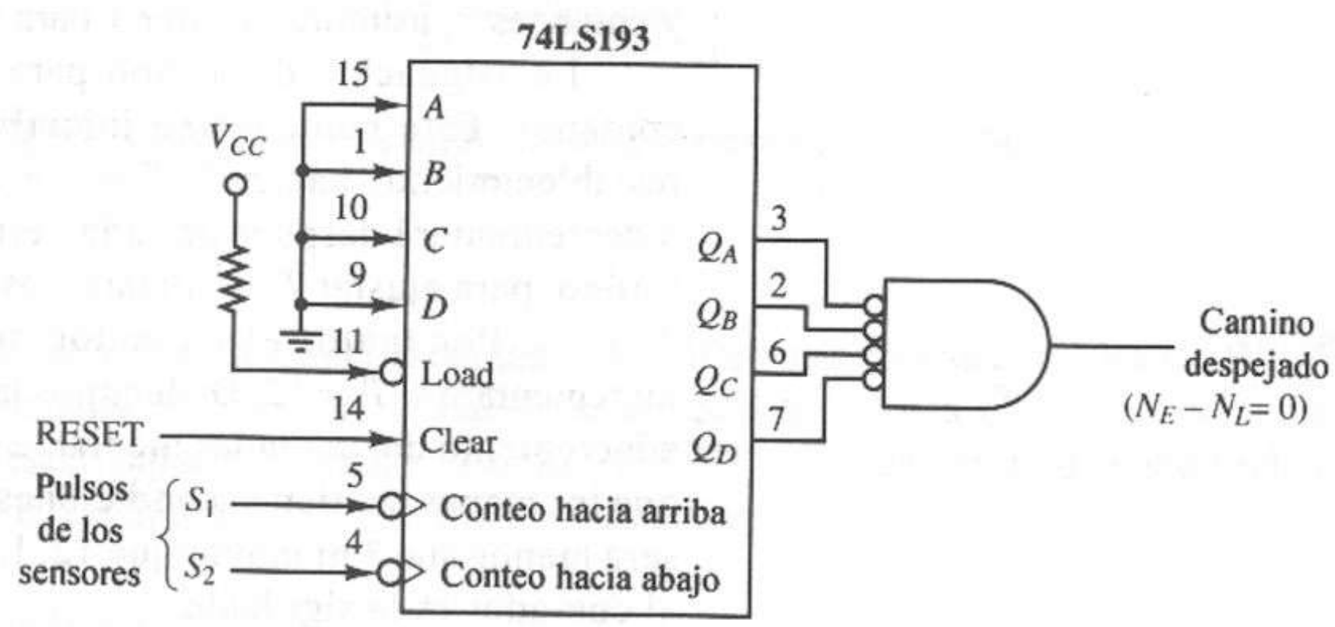


Figura 13.24 Diagrama lógico del contador de autos en el camino.

Contador de tráfico

El funcionamiento del contador de tráfico es similar al del contador de autos en el camino, pues mide la diferencia entre el número de autos que circulan en el camino en cada dirección. Utilizaremos de nuevo un contador arriba/abajo, incrementado por los autos que circulan en una dirección y decrementado por los autos que circulan en sentido opuesto. En este caso, sólo utilizaremos los pulsos de un sensor, S_1, y una señal de la unidad de control indicará la dirección del tráfico. Para minimizar la complejidad, supondremos que la diferencia entre el número de autos que circulan en el camino en las dos direcciones no será mayor que 15, por lo que basta un contador de 4 bits.

De nuevo utilizaremos un contador arriba/abajo 74LS193 de 4 bits, como se muestra en la figura 13.25. El contador se incrementa con cada pulso en $S1$ mientras $G1$ se encuentra activa, y se decrementa con cada pulso en $S1$ mientras $G2$ se encuentra activa. Restableceremos el contador cada 5 minutos.

Asignación de tiempo de luz verde

La cantidad total de tiempo asignado a las luces verdes en un ciclo de tráfico completo es

$$T_{tot} = T_1 + T_2$$

donde T_1 es el de tiempo asignado a la luz verde $G1$ en la dirección 1 y T_2 es el tiempo asignado a la luz $G2$ en la dirección 2. Si el tráfico en cierto periodo de 5 minutos es mayor en la dirección 1 que en la dirección 2, T_1 se incrementará en una unidad de tiempo y T_2 se reducirá en una unidad de tiempo, manteniendo T_{tot} constante. Para evitar una obstrucción del tráfico en cada dirección, no reduciremos el tiempo por debajo de un valor mínimo específico.

En este diseño asignaremos $T_{tot} = 160$ s, lo cual corresponde a 16 periodos de la señal de reloj CLK. Este tiempo se reparte entre T_1 y T_2. La figura 13.25 muestra el circuito. El contador de tiempo de luz verde 74LS93 se incrementa cada 10 segundos mientras está encendida alguna luz verde; es decir, mientras $G1 = 1$ o $G2 = 1$. La señal de reloj se desactiva cuando $G1 = G2 = 0$. Suponemos que $G1$ se enciende a la cuenta de 0. Un comparador 74LS85 detectará la condición $t = T_1$, y en ese momento $G1$ se apagará y el contador se detendrá hasta que $G2$ se encienda. A continuación, contará hasta 15 ($Q_DQ_CQ_BQ_A = 1111$), y entonces T_2 asumirá el valor 1 para apagar $G2$ en la unidad de control.

La asignación de tiempo para T_1 estará determinada por un segundo contador. Este contador se inicializa con el valor 7 en el momento del restablecimiento, haciendo $T_1 = T_2 = 80$ s. Luego, el contador se incrementará o decrementará después de cada periodo de 5 minutos, según el contador de tráfico, para ajustar T_1. Utilizaremos un tiempo mínimo de 40 segundos para T_1 y T_2. Por tanto, el contador no se decrementará si $T_1 = 3$ y no se incrementará si $T_1 = 12$. Deducimos las condiciones para inhibir el decremento e incremento del contador mediante los mapas K de la figura 13.26. Observe que los mapas contienen condiciones prescindibles, ya que el contador nunca será menor que 3 ni mayor que 12. La expresión lógica utilizada para inhibir al contador es la siguiente:

$$\text{INHIBIT} = DN \cdot (\bar{Q}_D\bar{Q}_C) + \overline{DN} \cdot (Q_DQ_C)$$

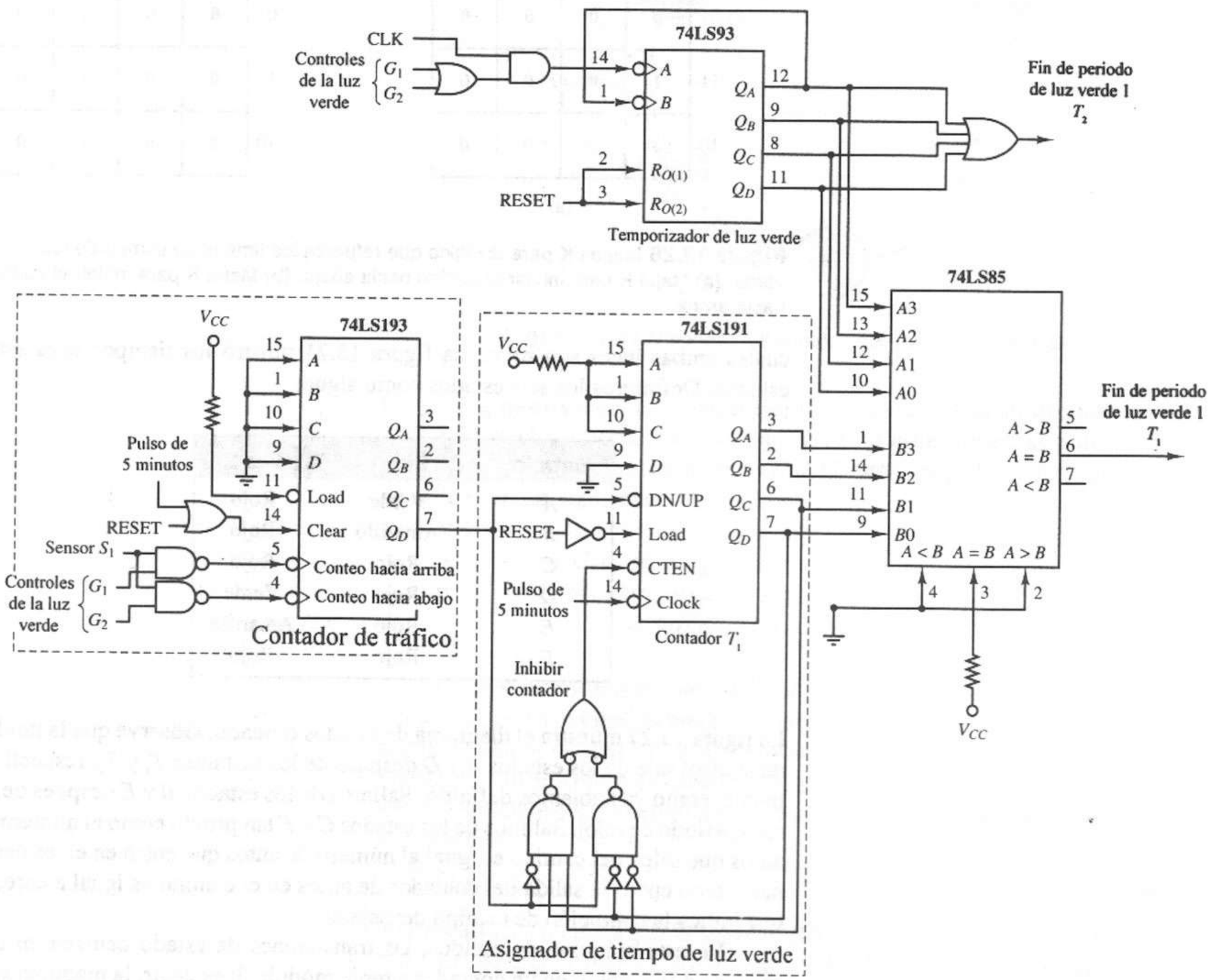


Figura 13.25 Diagrama lógico del temporizador de luz VERDE.

donde *DN* es la señal del contador de tráfico que controla la entrada $DN/\overline{UP}$ del contador T_1. La señal INHIBIT se aplica a la entrada $\overline{CTEN}$ del contador T_1, desactivando el contador cuando INHIBIT = 1 y activándolo cuando INHIBIT = 0. La figura 13.25 muestra el circuito lógico.

Unidad de control

La unidad de control requiere seis estados, correspondientes a los momentos durante los cuales la luz es verde y amarilla en cada dirección y durante los

Q_BQ_A \ Q_DQ_C	00	01	11	10
00	*d*	0	0	0
01	*d*	0	0	0
11	1	0	0	0
10	*d*	0	0	0

(a)

Q_BQ_A \ Q_DQ_C	00	01	11	10
00	0	0	1	0
01	0	0	*d*	0
11	0	0	*d*	0
10	0	0	*d*	0

(b)

Figura 13.26 Mapas K para la lógica que refuerza los límites de tiempo de luz verde. **(a)** Mapa K para inhibir el conteo hacia abajo. **(b)** Mapa K para inhibir el conteo hacia arriba.

cuales ambas luces son rojas. La figura 13.21 mostró los tiempos para estos estados. Definimos los seis estados como sigue:

Estado	Luz 1	Luz 2
A	Verde	Rojo
B	Amarillo	Rojo
C	Rojo	Rojo
D	Rojo	Verde
E	Rojo	Amarillo
F	Rojo	Rojo

La figura 13.27 muestra el diagrama de estados deseado. Observe que la unidad de control sale de los estados *A* y *D* después de los instantes T_1 y T_2, respectivamente, como ya habíamos definido. Salimos de los estados *B* y *E* después de un solo periodo de reloj. Salimos de los estados *C* y *F* tan pronto como el número de autos que salen del camino es igual al número de autos que entró en él, es decir, tan pronto como la salida del contador de autos en el camino es igual a cero, lo que indica la condición de camino despejado.

En esta máquina de estados, las transiciones de estado ocurren en una secuencia fija, como en un contador simple módulo 6; es decir, la máquina sólo realiza un ciclo a través de los estados *A–B–C–D–E–F–A*, etcétera. El momento de los cambios de estado depende de las tres entradas T_1, T_2 y Despejado.

Podemos utilizar varios métodos para diseñar esta máquina de estados; uno de ellos consistiría en diseñar un contador módulo 6, con un decodificador, para deducir las seis salidas. El contador se incrementaría con cada cambio de estado. Otra alternativa es desarrollar el diseño a partir de una tabla de estados de seis filas y ocho columnas, correspondientes a los seis estados y las tres entradas. Esta implantación requeriría tres *flip–flops* y varios circuitos de lógica combinatoria.

Para este ejemplo, utilizaremos una asignación de estados *one-hot*, definida en el capítulo 8, y realizaremos la máquina de estados con un registro de corrimiento de seis bits, como en la figura 13.28. Cada salida del registro de corrimiento corresponde a un estado de la máquina. Las salidas *A* y *B* controlan las luces

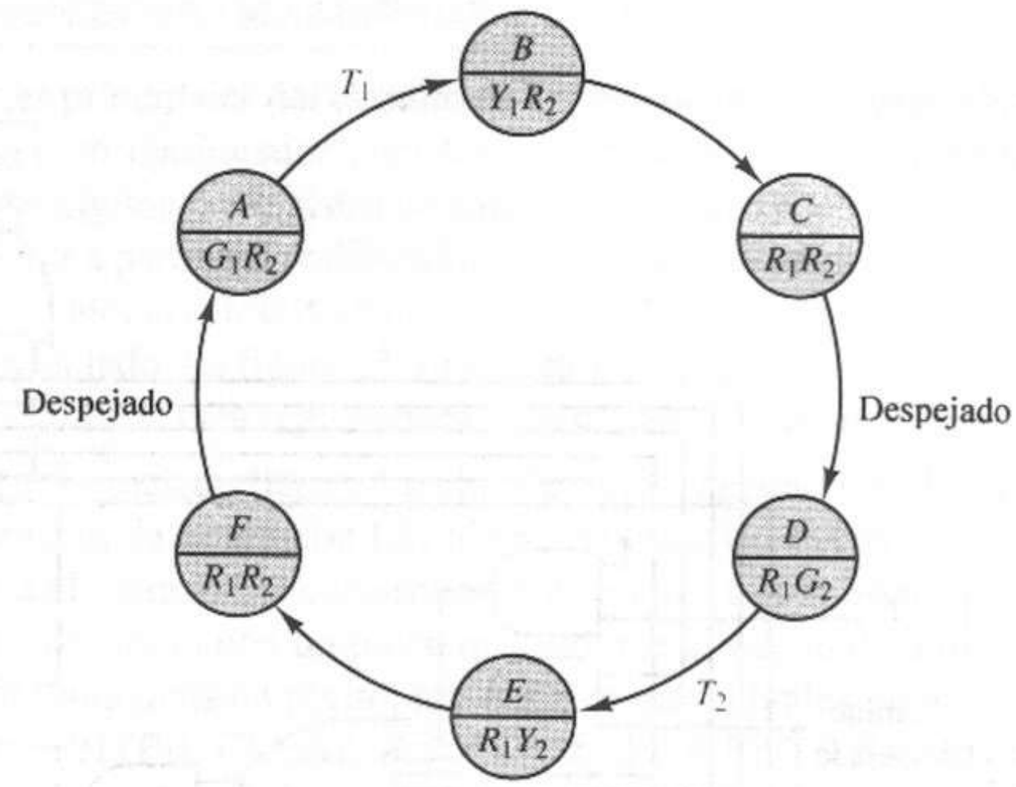


Figura 13.27 Diagrama de estados del controlador de tráfico.

$G1$ y $Y1$, respectivamente, mientras que las salidas D y E controlan las luces $G2$ y $Y2$. La luz $R1$ se enciende siempre que $G1$ y $Y1$ están apagadas y, de la misma manera, $R2$ se enciende siempre que $G2$ y $Y2$ están apagadas. Estas condiciones de salida son las siguientes:

$$\begin{aligned} G_1 &= Q_A & G_2 &= Q_D \\ Y_1 &= Q_B & Y_2 &= Q_E \\ R_1 &= \overline{(G_1 + Y_1)} & R_2 &= \overline{(G_2 + Y_2)} \end{aligned}$$

Cuando se oprime el botón RESET, el bit 0 del registro de corrimiento se inicializa con 1 y los demás bits con 0 para que la máquina inicie en el estado A. A continuación, se activa la entrada de activación del corrimiento y el registro realiza un corrimiento para cada condición indicada en el diagrama de estados. Estas condiciones se combinan para dar la siguiente señal de activación del corrimiento:

$$SHIFT_EN = (A \cdot T_1) + B + (C \cdot CLR) + (D \cdot T_2) + E + (F \cdot CLR)$$

La figura 13.28 muestra que la señal SHIFT_EN se combina mediante un AND con CLK para controlar las dos entradas $CK1$ del 74LS95, que proporciona la señal de reloj durante las operaciones de corrimiento. CLK también controla las entradas $CK2$ del 74LS95, que suministran la señal de reloj al registro durante las operaciones de carga.

13.4 Caja registradora de una tienda de abarrotes

La mayor parte de los establecimientos de venta al menudeo, incluidas las tiendas de abarrotes, utilizan cajas registradoras para cobrar a sus clientes. Las funciones básicas de una caja registradora en una tienda de abarrotes son introducir y mostrar los precios de los artículos individuales que el cliente desea adquirir, para después calcular y presentar la cuenta total. Muchas terminales electrónicas modernas de punto de venta calculan también descuentos, llevan un registro de los inventarios, calculan el cambio y realizan algunas otras funciones. Se utilizan

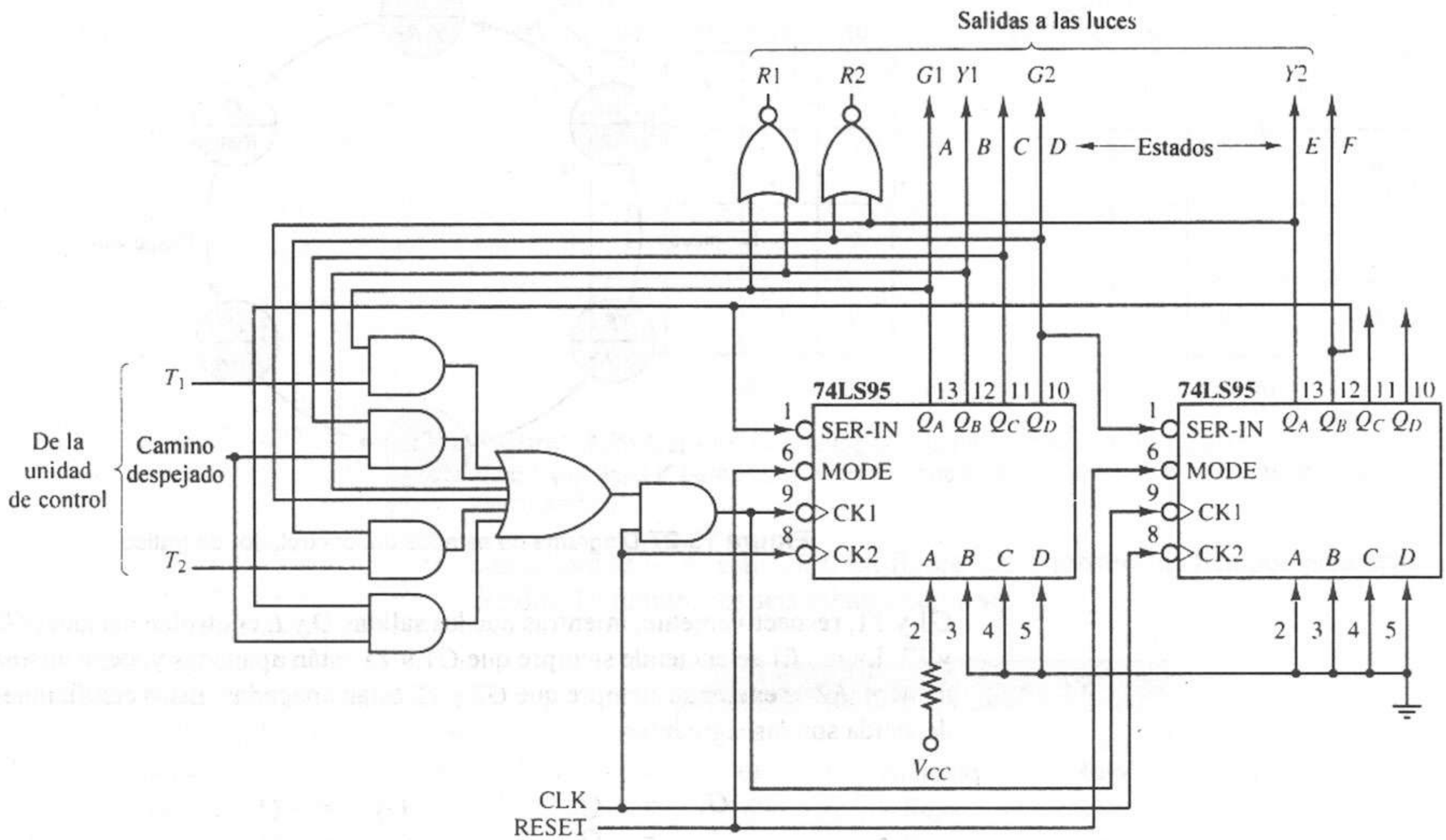


Figura 13.28 Diagrama lógico de la unidad de control del semáforo.

lectores magnéticos de código de barras o teclados para introducir los precios, el número de inventario, los descuentos y demás información.

Para este ejercicio, diseñaremos el circuito de control de una caja registradora con funciones mínimas, que sólo calcule la cuenta para una lista de artículos. Los precios de todos los artículos se introducirán mediante un teclado. La parte frontal se verá como el diagrama de la figura 13.29, con 14 botones y un *display* numérico. Los 14 botones incluyen los dígitos 0 a 9, y las siguientes funciones especiales:

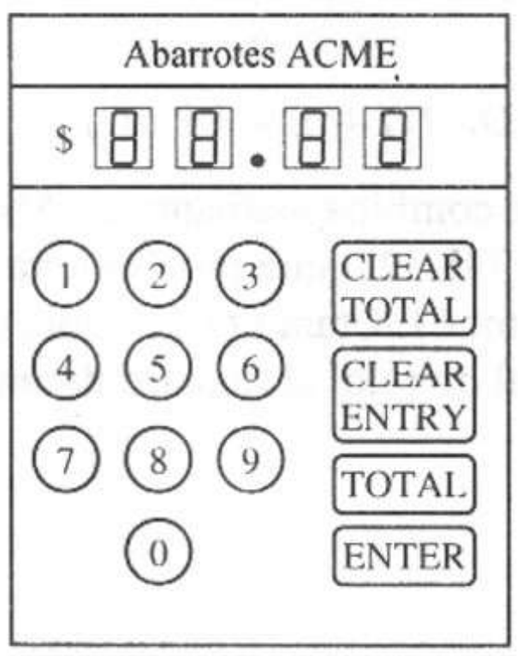


Figura 13.29 Caja registradora de una tienda de abarrotes.

ENTER: Se oprime después de teclear cada precio.

TOTAL: Se oprime después de haber introducido todos los precios, para mostrar la cuenta total.

CLEAR ENTRY: Se oprime para limpiar el *display* si se ha cometido un error al introducir el precio de un artículo.

CLEAR TOTAL: Se oprime para limpiar el total de la caja registradora antes de introducir los artículos de un nuevo cliente.

Para que el proyecto sea manejable, restringiremos los números a cuatro dígitos decimales, de modo que todos los precios y totales estarían en el intervalo [$00.01...$99.99]. Exhibiremos todos los números como valores decimales en *displays* LED de siete segmentos.

13.4.1 Necesidades del sistema

Las componentes principales del circuito de control de la caja registradora incluyen un teclado y un codificador para las 14 teclas de la caja registradora, un *display* de cuatro dígitos, un registro de entrada en el que se recorren los códigos BCD de cuatro bits a partir del codificador del teclado y se envían al *display*, y un acumulador, que calcula el total actual. Todos los elementos son controlados por señales desde el teclado. La figura 13.30 muestra un diagrama de bloques para el circuito de control de la caja registradora. Los componentes son los siguientes:

1. **Teclado.** El teclado tiene 14 teclas de botón ya descritas. Utilizaremos interruptores de botón con filtrado de rebotes. Un circuito codificador generará el valor BCD correspondiente a cada dígito numérico introducido, junto con un pulso que indique el hecho de que se ha oprimido una tecla de dígito. Las señales individuales de los botones TOTAL, ENTER, CLEAR TOTAL y CLEAR ENTRY se enviarán directamente a los registros de entrada y acumulador.
2. ***Display.*** Los dígitos decimales se presentarán en cuatro elementos LED de siete segmentos. Como los números deben aparecer en decimal, utilizaremos decodificadores BCD a siete segmentos para controlar el *display*. Durante la introducción del precio, los dígitos se recorrerán de derecha a izquierda a través del *display* al ser introducidos. Para cualquier número que necesite menos de cuatro dígitos no exhibiremos los ceros a la izquierda; es decir, estos dígitos aparecerán en blanco en el *display* para mejorar la legibilidad del número desplegado.
3. **Registro de entrada.** El registro de entrada se cargará con números introducidos desde el teclado y se recorrerá de derecha a izquierda hasta que se oprima la tecla ENTER. Las salidas del registro de entrada se enviarán al *display* y al acumulador, donde se calculará la cuenta. Si se introducen más de cuatro dígitos, sólo se conservarán los últimos cuatro. Después de oprimir la tecla ENTER, el valor se conservará en el registro de entrada, de modo que podamos añadir otro artículo del mismo precio a la cuenta con sólo oprimir la tecla ENTER de nuevo. Al oprimir la tecla TOTAL, el registro de entrada

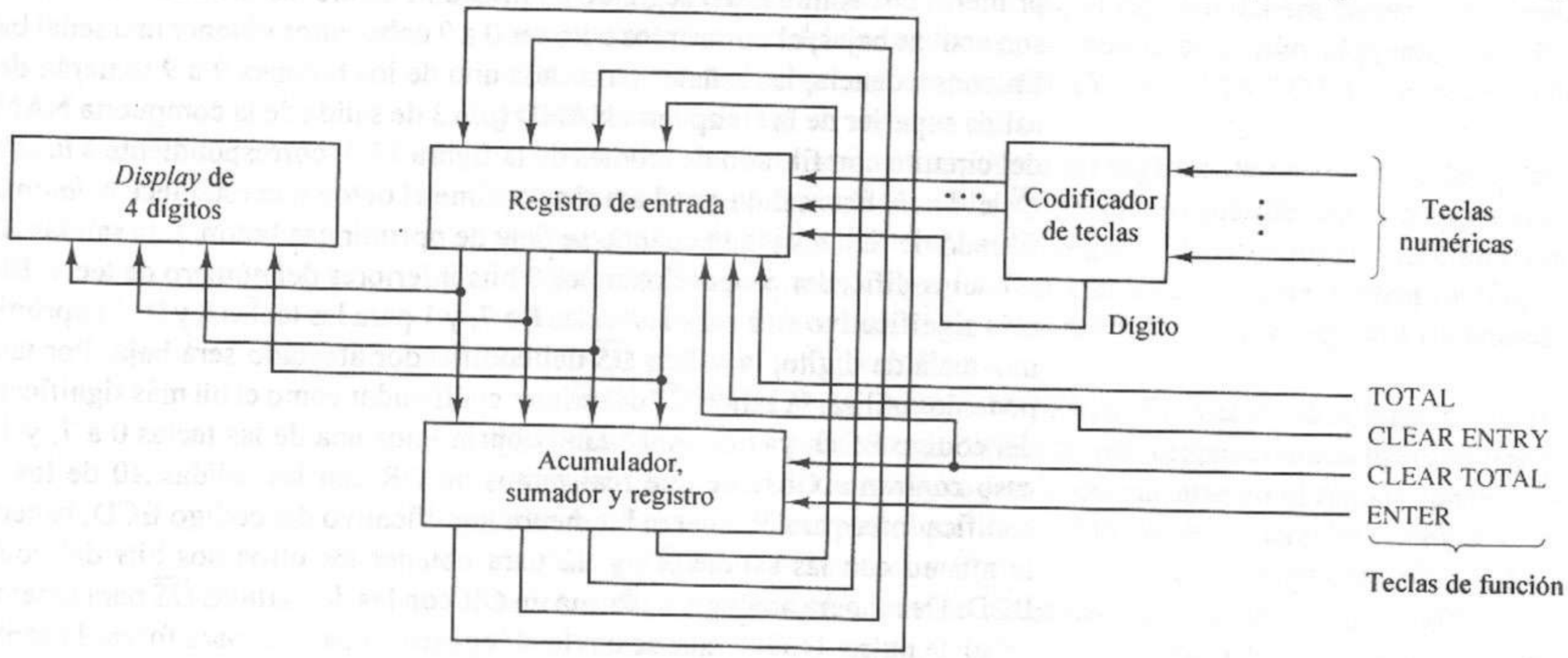


Figura 13.30 Diagrama de bloques de la caja registradora.

se cargará con la cuenta total del acumulador. Si se comete un error al introducir un precio, el registro de entrada se limpiará oprimiendo la tecla CLEAR ENTRY. El registro de entrada también se limpiará cuando se oprima la tecla CLEAR TOTAL en preparación para el próximo cliente.

4. **Acumulador.** El acumulador mantiene el total hasta ese momento de la venta actual. Para iniciar una nueva operación con un cliente, el acumulador se limpia oprimiendo la tecla CLEAR TOTAL. Posteriormente, cada vez que se oprima la tecla ENTER el precio que está en el registro de entrada se sumará al total actual que está en el acumulador. Como todos los números son decimales, utilizaremos un sumador BCD de cuatro dígitos para calcular los totales.

13.4.2 Diseño lógico

Como en nuestros ejemplos anteriores, realizaremos el diseño del circuito de la caja registradora diseñando y probando por separado cada uno de los módulos ya descritos. A continuación, conectaremos estos módulos entre sí y los probaremos hasta que todo el sistema funcione. Los siguientes párrafos presentan los diseños de los módulos individuales.

Teclado

Cada uno de los 14 botones del teclado será un interruptor de botón con filtrado de rebotes, como vimos antes (Fig. 13.3). Las señales producidas por las 10 teclas de dígitos numéricos (0 a 9) se codificarán en un valor BCD de 4 bits y una señal DIGIT enviada al registro de entrada para indicar que se ha oprimido una tecla de dígito. Las señales de las cuatro teclas especiales de función se enviarán de manera directa a los registros de entrada y acumulador para inicializar las operaciones correspondientes.

Produciremos los códigos de dígito BCD mediante un codificador 10 a 4 con prioridad construido como en la figura 13.31. El codificador 10 a 4 se construirá con dos codificadores 8 a 3 con prioridad 74LS148. Los botones 0 a 7 controlan las entradas del primer codificador, y los botones 8 y 9 controlan las primeras dos entradas del segundo codificador. Como las entradas del 74LS148 son activas bajas, al oprimir los botones 0 a 9 deberemos obtener una señal baja. En consecuencia, las señales para cada uno de los botones 0 a 9 tomarán de la salida superior de la compuerta NAND (*pin* 3 de salida de la compuerta NAND) del circuito con filtrado de rebotes de la figura 13.3, correspondiente a la salida $\overline{Q}$ del *latch*. Esta salida será baja si se oprime el botón que establece el *latch* con filtrado de rebotes y alta cuando se deje de oprimir ese botón. Las salidas $A2$ a $A0$ del codificador proporcionan los 3 bits inferiores del número de tecla. El bit más significativo es 0 para las teclas 0 a 7, y 1 para las teclas 8 y 9. Al oprimirse una tecla de dígito, la salida $\overline{GS}$ del codificador afectado será baja. Por tanto, podemos utilizar la salida $\overline{GS}$ del primer codificador como el bit más significativo del código BCD, ya que es 0 cuando oprimimos una de las teclas 0 a 7, y 1 en caso contrario. Observe que realizamos un OR con las salidas $A0$ de los dos codificadores para obtener el bit menos significativo del código BCD; hacemos lo mismo con las salidas $A1$ y $A2$ para obtener los otros dos bits del código BCD. De manera análoga, se forma un OR con las dos salidas $\overline{GS}$ para tener una señal de pulso, DIGIT, que se envía al registro de entrada para forzar la captura de un nuevo dígito.

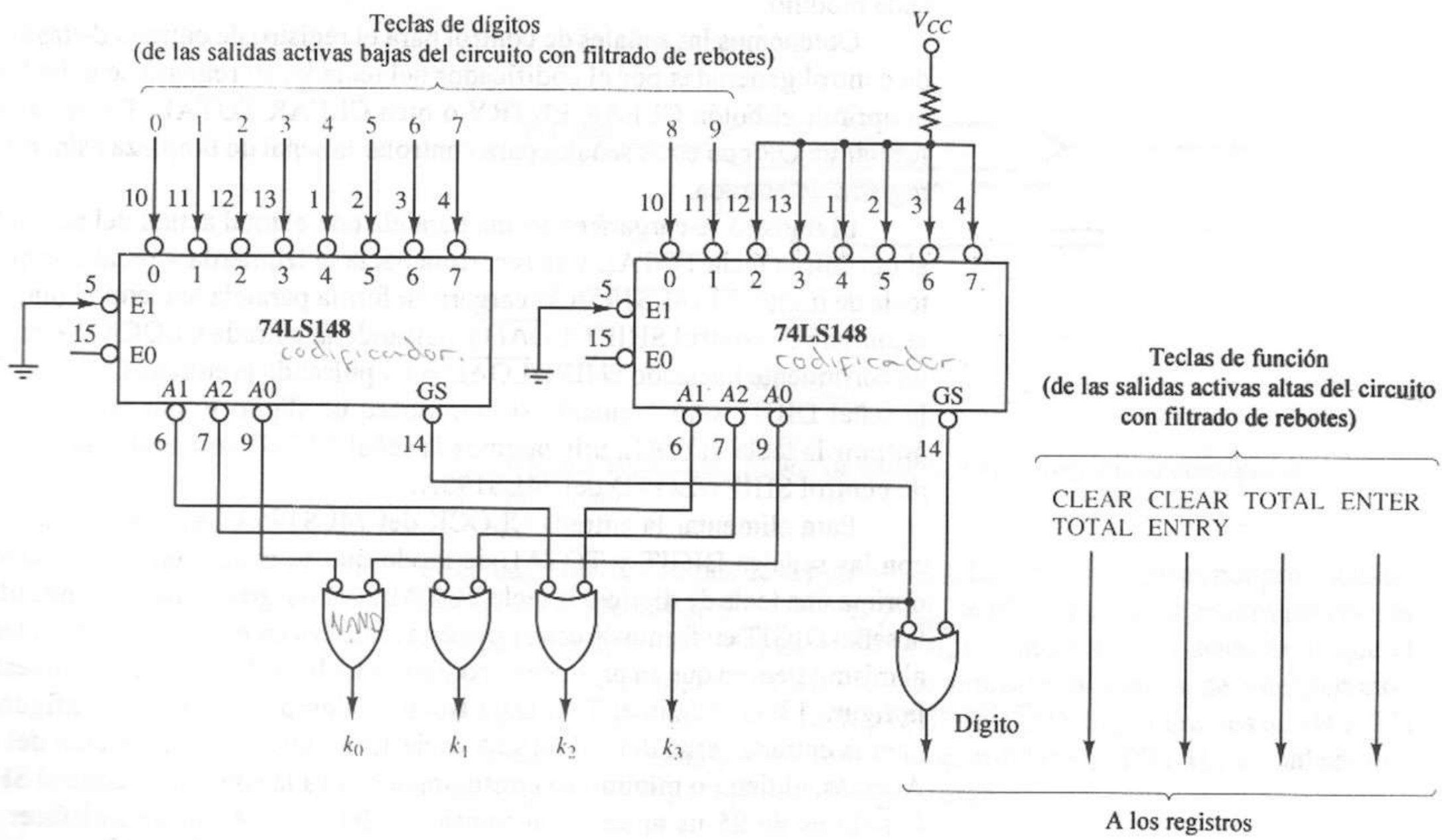


Figura 13.31 Codificador del teclado de la caja registradora.

Registro de entrada

El registro de entrada debe almacenar hasta cuatro dígitos para el *display*; estos dígitos también se envían al acumulador para calcular el precio total. A fin de crear el efecto del corrimiento de los dígitos de derecha a izquierda en el *display* durante la introducción de los precios, el registro deberá cargarse en forma serial con dígitos BCD del codificador del teclado cuando se reciba un pulso en la línea DIGIT. Al oprimir la tecla TOTAL, el registro deberá cargarse en forma paralela con el valor del acumulador para enviar al *display* el precio total. Por último, al oprimir el botón CLEAR ENTRY o CLEAR TOTAL, el registro de entrada se inicializará en ceros.

Las funciones anteriores requieren un registro de corrimiento de 16 bits que realice las funciones de corrimiento hacia la izquierda, carga en paralelo y de limpiar. El módulo 74LS195A es un registro de corrimiento de 4 bits, con corrimiento sincronizado, carga paralela y con entradas de control de limpiar asíncronas. Utilizaremos cuatro de estos módulos para el registro de entrada, como se muestra en la figura 13.32.

Cada dígito BCD es un valor de 4 bits. En vez de desplazar un nuevo dígito hacia el registro de entrada 1 bit a la vez, organizaremos los módulos de registro de modo que todo el dígito pueda desplazarse en el registro en un solo paso. La figura 13.32 muestra que el 74LS195A de la derecha contiene el bit menos significativo de los cuatro dígitos del *display*; el siguiente 74LS195A a su izquierda contendrá el siguiente bit de cada uno de los cuatro dígitos, y así sucesivamente. De este modo, en un corrimiento a la izquierda los 4 bits correspondientes al dígito de la extrema izquierda del *display* se extraerán de

los módulos del registro mientras se introducen los bits del nuevo dígito, uno en cada módulo.

Obtenemos las señales de control para el registro de entrada de las señales de control generadas por el codificador del teclado. El registro se debe limpiar al oprimir el botón CLEAR ENTRY o bien CLEAR TOTAL. Por tanto, basta formar un OR con estas señales para controlar la señal de limpieza asíncrona del registro de entrada.

El registro se cargará en forma paralela con el total actual del acumulador al oprimir la tecla TOTAL y se recorrerá hacia la izquierda 1 bit al oprimir una tecla de dígito. El 74LS195A se cargará en forma paralela haciendo igual a cero la entrada de control SHIFT/$\overline{\text{LOAD}}$ y pulsando la entrada CLOCK. Se realizará un corrimiento haciendo SHIFT/$\overline{\text{LOAD}}$ = 1 y pulsando la entrada CLOCK. Como la señal DIGIT será 1 cuando se introduzca un dígito y, por tanto, será 0 al oprimir la tecla TOTAL, utilizaremos la señal DIGIT para controlar la entrada de control SHIFT/$\overline{\text{LOAD}}$ del 74LS195A.

Para alimentar la entrada CLOCK del 74LS195A hay que formar un OR con las señales DIGIT y TOTAL de modo que se genere un pulso cuando se oprima una tecla de dígito o la tecla TOTAL. Por desgracia, no podemos utilizar la señal DIGIT en forma directa, pues ésta se activa en el controlador del teclado al mismo tiempo que se produce el código de tecla de 4 bits, como se muestra en la figura 13.33. Según el TTL Data Book, el tiempo mínimo de configuración para la entrada serial del 74LS195A es de 15 ns antes de la transición del reloj. Además, el tiempo mínimo de configuración para la entrada de control SHIFT/$\overline{\text{LOAD}}$ es de 25 ns antes de la transición del reloj. A fin de satisfacer estos requisitos, será necesario retardar el pulso sobre la señal DIGIT. Produciremos este retardo haciendo pasar la señal DIGIT a través de una cadena de inversores, como se muestra en la figura 13.32, lo cual genera la señal CLOCK que aparece

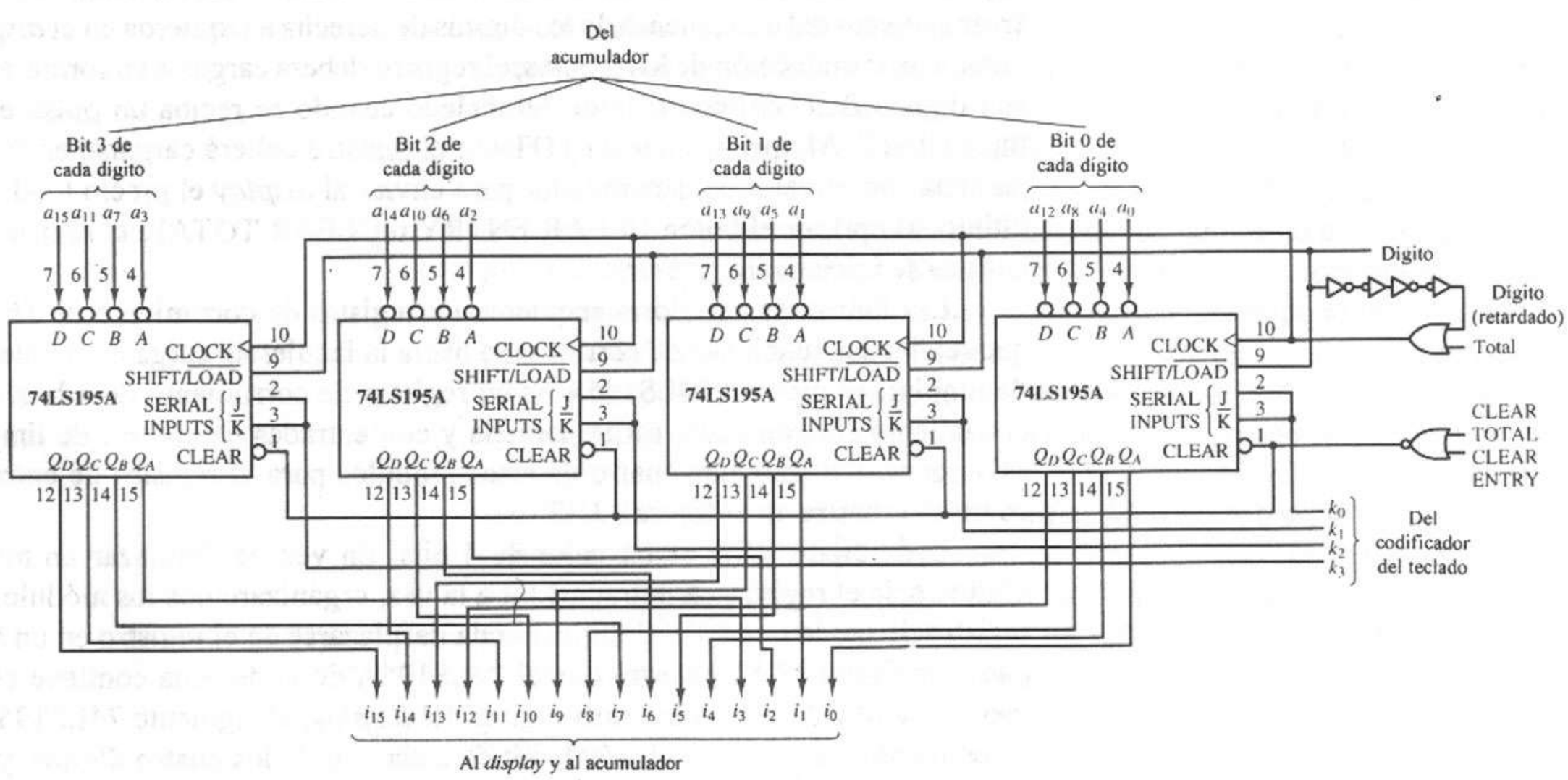


Figura 13.32 Registro de entrada.

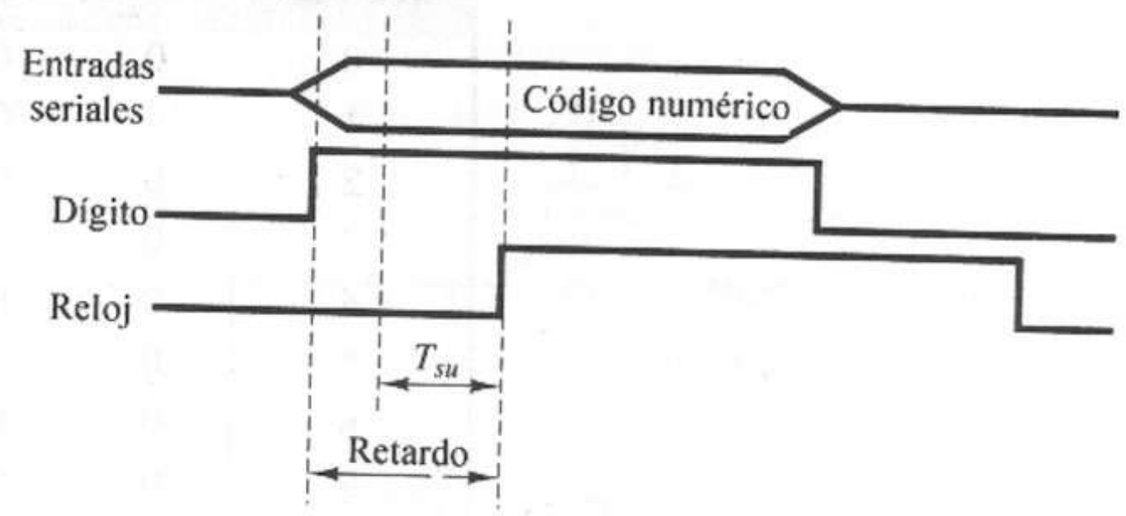


Figura 13.33 Distribución de tiempos para el registro de corrimiento de entrada.

en el diagrama de tiempos de la figura 13.33. Los retardos típicos por propagación del inversor 74LS04 son $T_{PLH} = 9$ ns y $T_{PHL} = 10$ ns. Por tanto, cuatro inversores retardarán la señal del reloj en un valor típico de 38 a 40 ns después de que el pulso DIGIT sea alto, lo cual satisface los requisitos de tiempo de configuración para las entradas de control CLOCK y SHIFT/$\overline{\text{LOAD}}$. Supondremos que la salida del acumulador es estable mucho antes de oprimir la tecla TOTAL, lo cual permite utilizar la señal TOTAL para el reloj de carga.

Acumulador

El acumulador comprende un registro de carga paralela de 16 bits y un sumador BCD de cuatro dígitos. El acumulador se limpia cuando se oprime el botón CLEAR TOTAL y al oprimirse la tecla ENTER se carga con la suma de su contenido y el número que está en el registro de entrada.

Implantaremos el registro del acumulador con dos módulos de *flip-flop* D octales 74LS273, como se muestra en la figura 13.35. Las entradas de reloj son controladas por la señal ENTER del teclado, y las entradas de limpieza son controladas por la señal CLEAR TOTAL del teclado.

Los sumadores decimales codificados en binario no están disponibles como módulos TTL estándar. Por tanto, tendremos que crear uno a partir de sumadores binarios. Consideremos la suma de dos dígitos BCD con un sumador binario de 4 bits. La suma de dos dígitos decimales será un número en el intervalo [0...18]. La tabla 13.4 enumera las salidas de suma y acarreo producidas por un sumador binario cuando sus entradas son dígitos BCD, junto con las salidas de suma y acarreo correspondientes deseadas del sumador BCD. Podemos identificar tres casos únicos en esta tabla.

Caso 1: $0 \leq$ suma ≤ 9. En este caso, los resultados obtenidos por el sumador binario son idénticos a los necesarios para el sumador BCD. Por tanto, podemos utilizar los resultados sin ajustes.

Caso 2: $10 \leq$ suma ≤ 15. En este caso, el sumador binario produce sumas en el intervalo $(1010)_2 \ldots (1111)_2$, sin salida de acarreo. Podemos obtener los resultados decimales correspondientes sumando $6_{10} = (0110)_2$ a la salida del sumador binario. Esta operación produce una salida de acarreo y una suma en el intervalo $[(0000)_2 \ldots (0101)_2]$, como se deseaba.

TABLA 13.4 SUMA BINARIA CONTRA BCD

Suma decimal	Sumador binario		Sumador BCD		Ajuste de binario a BCD
	C_{out}	Suma	C_{out}	Suma	
0	0	0000	0	0000	+0000
1	0	0001	0	0001	+0000
2	0	0010	0	0010	+0000
3	0	0011	0	0011	+0000
4	0	0100	0	0100	+0000
5	0	0101	0	0101	+0000
6	0	0110	0	0110	+0000
7	0	0111	0	0111	+0000
8	0	1000	0	1000	+0000
9	0	1001	0	1001	+0000
10	0	1010	1	0000	+0110
11	0	1011	1	0001	+0110
12	0	1100	1	0010	+0110
13	0	1101	1	0011	+0110
14	0	1110	1	0100	+0110
15	0	1111	1	0101	+0110
16	1	0000	1	0110	+0110
17	1	0001	1	0111	+0110
18	1	0010	1	1000	+0110

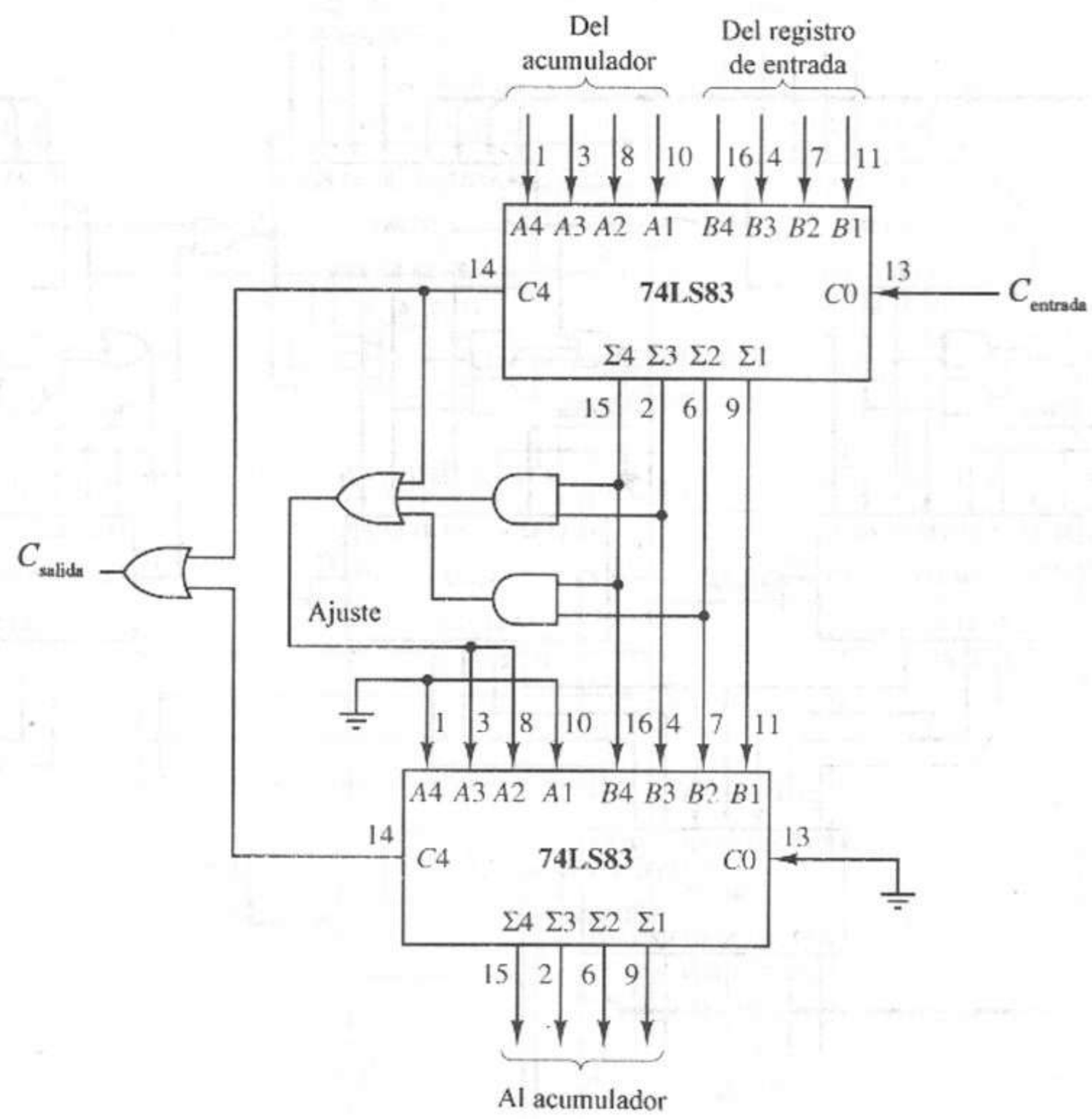
Caso 3: 16 ≤ suma ≤ 18. En este caso, los sumadores binario y BCD producen un acarreo, de modo que la suma binaria es 6 unidades menor que el resultado BCD deseado. Obtenemos los resultados decimales correctos sumando de nuevo $6_{10} = (0110)_2$ a la salida del sumador binario, como en el caso 2.

En los casos 2 y 3, el sumador decimal genera un acarreo de salida para cualquier suma mayor o igual que 10_{10}, mientras que el sumador binario genera un acarreo de salida sólo si la suma es mayor o igual que 16_{10}. En consecuencia, si sumamos 6 a la salida del sumador binario para sumas mayores que 9_{10}, los resultados se ajustarán al valor deseado.

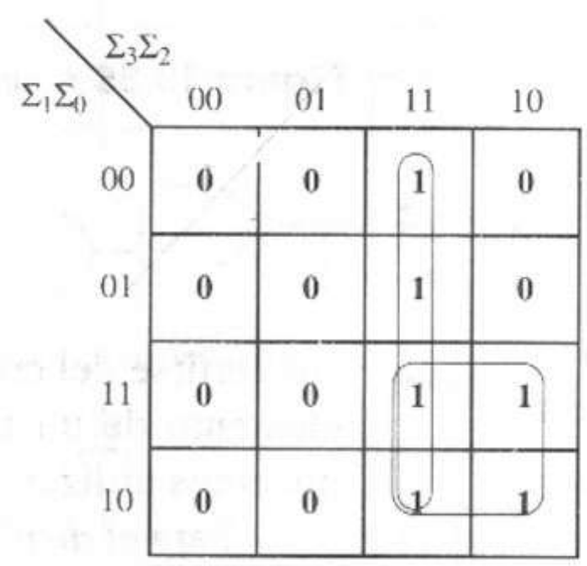
La figura 13.34a muestra un circuito con dos etapas que suma dos dígitos BCD examinando la salida del primer sumador binario y ajustando el resultado sumando 6 a éste si la suma está en el intervalo $[(1010)_2 ... (1111)_2]$ (caso 2) o si la salida del acarreo es 1 (caso 3). Utilizaremos el mapa K de la figura 13.34b, el cual asocia valores de la suma mayores que 9, de modo que la condición que indica la necesidad de ajustar el resultado es

$$Adjust = C_{out} + \Sigma_3 \Sigma_2 + \Sigma_3 \Sigma_1$$

Utilizaremos dos módulos sumadores de cuatro bits 74LS83, uno para calcular la suma y otro para ajustar el resultado. Si *Adjust* = 1, sumamos $(0110)_2$ a la



(a)



$\Sigma_1\Sigma_0$ \ $\Sigma_3\Sigma_2$	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	1

(b)

Figura 13.34 Sumador decimal codificado en binario de un dígito. **(a)** Diagrama lógico. **(b)** Mapa K de una suma ≥ 10.

salida del primer sumador; en caso contrario, sumamos $(0000)_2$. La figura 13.35 presenta el circuito acumulador de 16 bits completo.

Display

El *display* de la caja registradora de cuatro dígitos es similar a los dos *displays* de tres dígitos diseñados anteriormente para el juego de la máquina tragamonedas. Para el *display* de la caja registradora, cada uno de los cuatro dígitos BCD debe

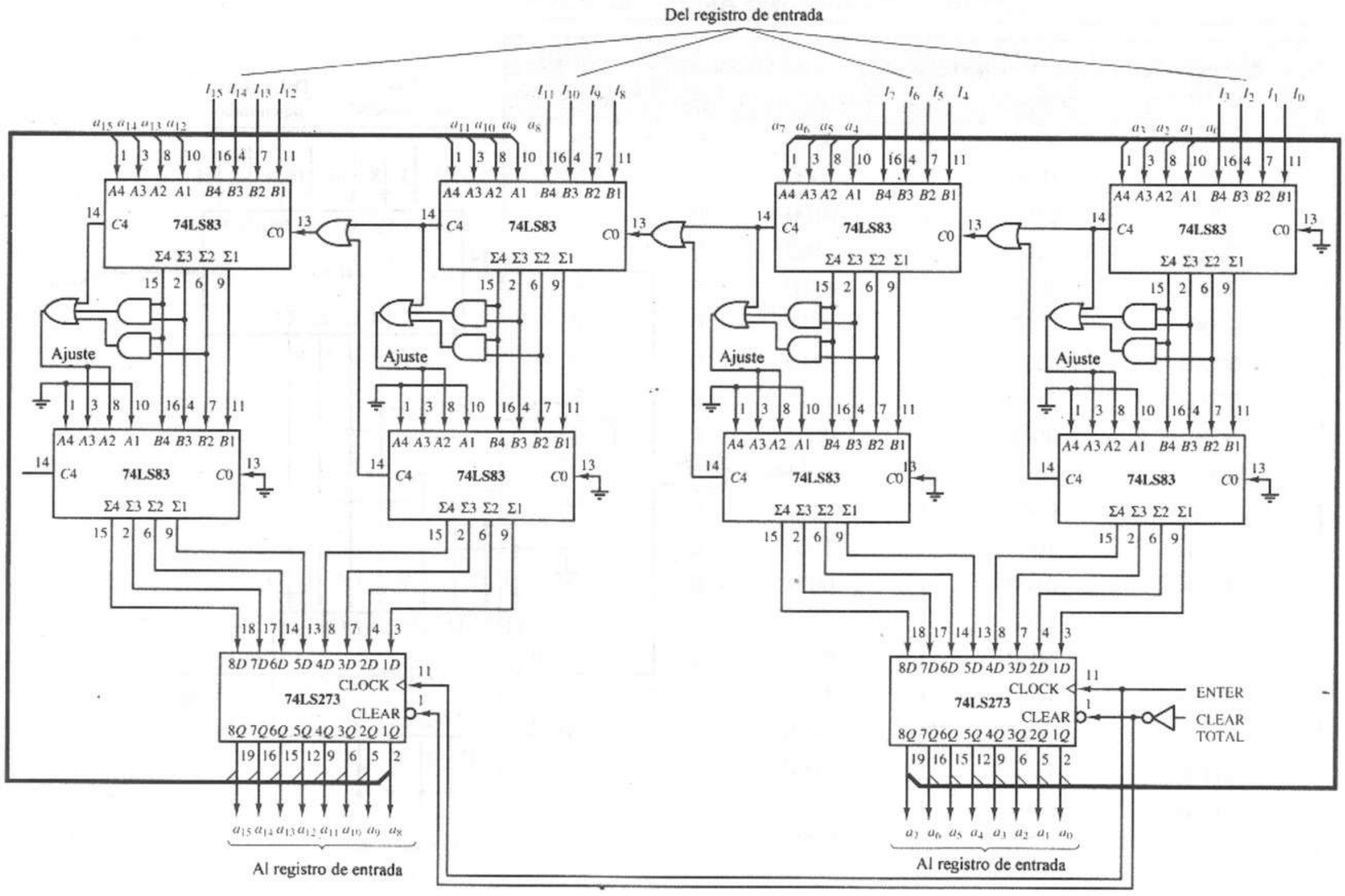


Figura 13.35 Acumulador decimal codificado en binario.

convertirse del código BCD al código de siete segmentos para exhibirlo como elemento de un LED de siete segmentos. Como con los diseños anteriores, podemos utilizar un módulo convertidor de código 74LS48 con ese fin.

Para el *display* de la caja registradora también necesitamos que los dígitos correspondientes a ceros a la izquierda aparezcan en blanco en el *display*; es decir, si un número comienza con uno o más ceros, estos elementos deberán aparecer en blanco en el *display*. El 74LS48 incluye una entrada de control BI/RBO la cual, cuando es igual a 0, hace que los siete segmentos de LED estén apagados.

Para esta aplicación, si los bits 15 a 12 del registro de entrada son 0000, la entrada BI/RBO del 74LS48 más significativo deberá asumir el valor 0 para poner en blanco el elemento más significativo del *display*. El siguiente dígito, correspondiente a los bits 11 a 8 del registro de entrada, queda en blanco si y sólo si los bits 11 a 8 son 0000 y el dígito más significativo también está en blanco. De manera similar, los elementos controlados por los bits 7 a 4 y los bits 3 a 0 del registro de entrada quedarán en blanco si y sólo si todos los dígitos superiores son iguales a 0.

La figura 13.36 muestra el circuito final para el *display*.

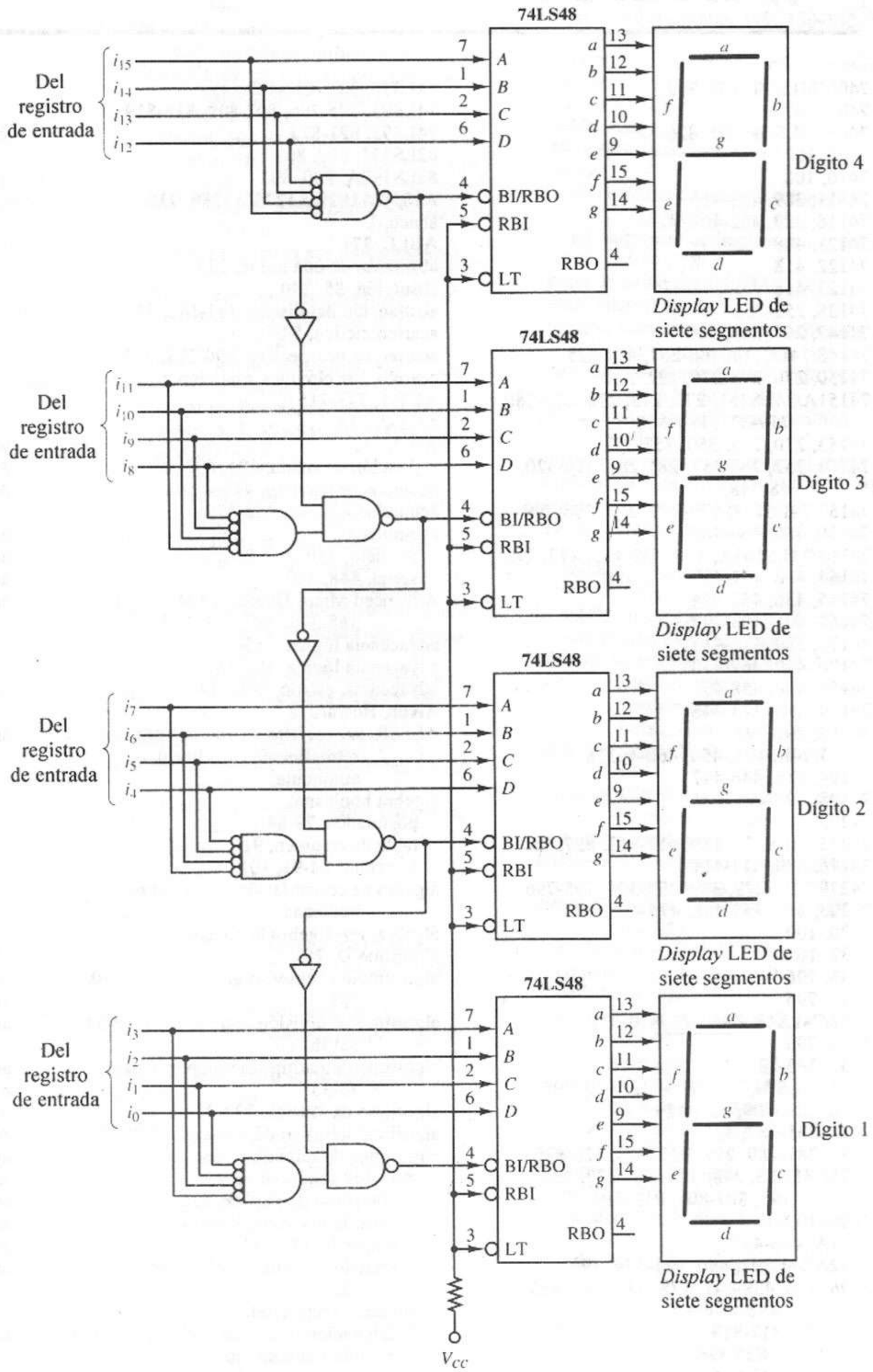


Figura 13.36 *Display* de la caja registradora con control de espacios en blanco.